SeNDORComm: An Energy-Efficient Priority-Driven Communication Layer for Reliable Wireless Sensor Networks

Vinaitheerthan Sundaram, Saurabh Bagchi, Yung-Hsiang Lu, Zhiyuan Li*

School of Electrical and Computer Engineering, Department of Computer Science (*)
Purdue University, West Lafayette, IN 47907
{vsundar,sbagchi,yunglu,li}@purdue.edu
Contact Author: Saurabh Bagchi

Abstract

In many reliable Wireless Sensor Network (WSN) applications, messages have different priorities depending on urgency or importance. For example, a message reporting the failure of all nodes in a region is more important than that for a single node. Moreover, traffic can be bursty in nature, such as when a correlated error is reported by multiple nodes running identical code. Current communication layers in WSNs lack efficient support for these two requirements. We present a priority-driven communication layer, called SeNDORComm, which schedules transmission of packets driven by application-specified priority, buffers and packs multiple messages in a packet, and honors the latency guarantee for a message. We show that SeNDORComm improves energy efficiency, message reliability, and network utilization and delays congestion in a network. We extensively evaluate SeNDORComm using analysis, simulation, and testbed experiments. We demonstrate the improvement in goodput of SeNDORComm over the default communication layer, GenericComm in TinyOS (134.78% for a network of 20 nodes).

Keywords: *Wireless sensor network, reliable message delivery, congestion avoidance, priority driven communication, message aggregation.*

1. Introduction

In many WSN applications certain events are more important than others and therefore messages reporting such events are more urgent. For example, in surveillance applications [6], the alert sent for an intruding pedestrian is less urgent than that sent for an intruding motor vehicle. Similarly, in an indoor climate control application, presence of harmful gas in the air should be reported more urgently than the current CO_2 level. Another class of events common in WSNs is run-time errors detected by a monitoring infrastructure such as in [8][[7][10]. The severity of errors is often different, which necessitates different priorities for the error messages. As an example, our recent error detection framework called H-SEND [9] distinguishes two kinds of error messages. The advisory error messages (e.g. messages indicating a transient interfering source went through the network) have much lower priority than severe error messages which need attention as soon as possible (e.g. messages indicating a fraction of nodes without a cluster head to align with).

In this paper, we seek to improve network utilization by leveraging the fact that different messages have different priorities for transmission. Thus, techniques used for network throughput enhancement [2][3][4], such as message combining or piggybacking, must honor message priorities to ensure that high-priority messages are delivered without long waits while preventing starvation of the lower priority messages. Motivated by these insights, we propose an energy-efficient priority-driven communication layer, called SeNDORComm, for reliable operation of WSNs. In this framework, we distinguish the messages with the highest priority that need immediate attention, called *immediate messages,* from the other messages, called *deferred messages*. SeNDORComm schedules transmission of packets driven by application-specified priority, buffers deferred messages and combines multiple deferred messages in a packet, and honors the latency guarantee for a message.

The error traffic in WSNs can be bursty. For example, a change in the environment can lead to multiple sensor nodes reporting a drop in throughput, which lead to a storm of error alerts traveling toward the base station. Due to limited bandwidth in WSNs, the possibility of congestion under bursty traffic is high. SeNDORComm handles bursty traffic efficiently by combining multiple messages into a packet based on priority.

Combining multiple messages into a packet based on priority has the following advantages: It conserves energy, reduces congestion by sending fewer packets (since there is a fixed overhead to each packet), and prioritizes the transmission of important messages. Although this functionality may be implemented at different layers—application, routing, or link layer—there are several compelling reasons for our choice to implement it in a new (communication) layer between the application and the routing layers. First, combining using priorities is potentially useful for all application components that generate messages and thus, it is preferable to provide the functionality beneath the application layer. On the other hand, although there are a few advantages to implement combining at the routing/link layers [2][3][4], several significant disadvantages make this option unattractive. At the routing/link layers, each packet has to be repacked multiple times, thus increasing delay and complexity at each intermediate hop. Repacking at each intermediate node will require additional information for each message such as priority and cumulative waiting time of this message in previous message queues, to be carried along with the message. Next, the priorities are essentially an end-to-end property of messages and therefore it is logical to not process them at intermediate hops. Finally, not touching the routing/link layer potentially allows more applications to take advantage of our solution.

To evaluate SeNDORComm, we first perform a queuing theory based analysis of SeNDORComm to determine an upper bound on the number of explicit packets generated as a function of the latency guarantee. We implement SeNDORComm in TinyOS and conduct experiments to show the energy and goodput advantage over the baseline (GenericComm, the default communication layer in TinyOS) for different levels of interference and loads. The results show a 59.3% reduction in energy for high interference condition. The goodput advantage of SeNDORComm over GenericComm for a highly-loaded network with 20 Mica2 motes is 134.78%, which illustrates SeNDORComm's ability to handle bursty traffic. For demonstrating scalability, we perform a simulation study with 100 nodes, which shows a 42% improvement in goodput and a 176.5% improvement in reliability of immediate messages under heavy load.

Summarizing, the key contributions of our paper are:

- We provide a communication layer for a WSN, called SeNDORComm, that handles prioritized messages while optimizing the energy overhead of transmission and its efficient implementation in Tiny OS.
- We show the benefits of combining priority and piggybacking in WSNs which include energy conservation, congestion avoidance, and capability to handle bursty traffic.

The rest of the paper is organized as follows. In Section 2 we discuss related work. The detailed design of SeNDORComm is presented in Section 3. The analysis and discrete-event simulations are explained in Section 4. Section 5 and 6 presents the results from the test bed and the simulation. A discussion of future work is in Section 7. Section 8 concludes the paper.

2. Related Work

We confine ourselves to literature in WSN since these are most relevant due to the domain specific challenges. We classify the related work into four categories: *(1) Protocols that use priorities, (2) Protocols that do message pooling, (3) Protocols that address congestion control, and* (4) *Investigation of message size on throughput.*

1. *Priorities*. RAP [1] is motivated toward real-time communication. It is an application layer that determines message priorities to control the latency of its journey to the base station. It could be structured on top of SeNDORComm which would perform the message transmissions according to the priorities. In [17] the authors present a protocol for sensor-actor coordination to control the quality of data collection. Actor interests may have different levels of importance and sensors arrange the use of their resources according to the importance of interests.

2. *Message pooling*. AIDA (Adaptive Application-independent Data Aggregation) [2] lies between the network and the data-link layer. It buffers messages from the network layer in an FCFS queue. It aggregates messages to different degrees depending on the MAC layer contention. Being at a lower layer, it does not concern itself with message priorities and the related issue of bounding latency for messages awaiting aggregation. The widely used CSMA-based MAC layer for WSNs called BMAC [4] provides a mechanism to amortize the cost of sending long preambles before each packet. If there are multiple packets destined to a receiver, once synchronization between

the sender and the receiver is established, BMAC can omit sending the preamble for all but the first packet. However, the application has to maintain a message pool destined to the same node and decide when to send the accumulated messages. SeNDORComm removes this burden from the application. SeNDORComm can benefit from BMAC's batching at each intermediate hop.

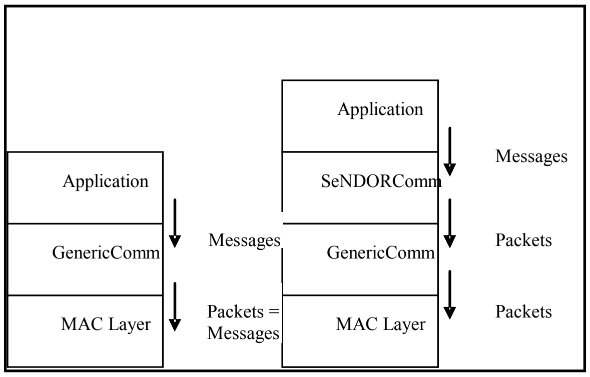
The Sensornet protocol (SP)[3] sits between the network and the link layer and provides a minimal interface needed so that different network protocols and different MAC-layer protocols can be plugged in. It supports the network layer indicating if a message is urgent in which case it is sent without pooling. In case it is not flagged as such, SP tries to batch multiple messages. Being lower down in the protocol stack, it does not concern itself with application priorities.

3. *Congestion control*. There exists a significant volume of work at the link layer to provide congestion avoidance or control [11][12]. The approach is to detect or anticipate congestion and prevent a congestion collapse by appropriately backing off communication. SeNDORComm can coexist with such techniques and also with hints from the application; it can further avoid a congestion collapse. Some work at the routing layer [19] distributes the traffic load to route around congestion.

4. *Message size and throughput*. One design decision of SeNDORComm is based on the fact that throughput improves with increasing message size as long as the channel losses do not outweigh the efficiencies. The impact of message size on throughput was first studied analytically and through simulation by Akyildiz *et al.* [13]. A datalink layer protocol called SEDA [18] shows that by reducing the granularity of retransmission to a smaller block rather than the entire MAC frame or packet, throughput can be improved even in lossy channels. This work is very germane to SeNDORComm. If SeNDORComm's decision to piggyback multiple messages in a packet increases the loss rate (we empirically show that we can operate in a conservative region where it does not), then it can use SEDA to recover the affected messages rather than retransmitting the entire packet.

3. SeNDORComm Design and Implementation

SeNDORComm is a communication layer that sits between the application and the network layer. The network layer interface in TinyOS is called GenericComm as it provides a generic interface to the application irrespective of underlying protocol layers. The radio stack for WSN applications that use GenericComm and SeNDORComm is shown in Figure 1. We designed SeNDORComm with the following design goals in mind: 1) reduce deferred message traffic to conserve energy, 2) send critical messages promptly, and 3) keep the interface simple and close to GenericComm, so existing TinyOS applications can be easily integrated. We explain how these design goals are met.



```
interface SeNDORSend {
  command result_t send( uint16_t  address, uint8_t  length,
                                        SC_TOS_MsgPtr  msg, uint8_t  urgency);
  event result_t sendDone( SC_TOS_MsgPtr msg, result_t  success );
}

interface SeNDORReceive {
  event  SC_TOS_MsgPtr receive( SC_TOS_MsgPtr   msgPtr );
  command  void  receiveDone( SC_TOS_MsgPtr   msgPtr );
}

interface SeNDORCommCtl {
  command  result_t  receiveInit( SC_TOS_MsgPtr  recvQ, uint8_t  size );
  command  result_t  pause(  );
  command  result_t  resume(  );
}
```

Figure 1. WSN application's radio stack interface respectively without and with SeNDORComm(left). SeNDORComm Interfaces: Send, Receive, Control(right).

3.1. Message Priorities and Queuing Policy

SeNDORComm allows the application to specify priorities for all messages. Since this is application-specific priority, it is conceivable that more than a few priority values are desired. The allowable range of priority values is the range of one-byte unsigned integer i.e., 0 to 255. Following the tradition of assigning lower values to denote higher priority, 0 denotes the highest priority (the immediate messages) and 255 denotes the lowest priority. In practice, we anticipate a smaller number of priority levels will be used (for our experiments we use three).

The policy for deciding when to send a message is at the heart of SeNDORComm's design. By our policy, the immediate messages are sent without buffering and the lowest priority messages are not sent and are logged in the local persistent storage for later retrieval. Deferred messages are messages with priority $\in$ (1, 254) and are buffered by SeNDORComm. The deferred messages are stored in a priority queue. For multiple messages with the same priority, they are unordered. Later, they are either piggybacked on other messages destined to the same node or sent out as a separate packet called an *explicit packet* as explained later. The guarantee from SeNDORComm is that a message with priority value i is always sent before or together with a message with priority value j, where j

> i. However, if the message with priority value j is in the process of being sent when the message with priority value i arrives, then the sending is completed. Thus, the queuing discipline is a non-preemptive priority queue.

3.3 Operation of SeNDORComm

SeNDORComm Send. When an application requests SeNDORComm to send an immediate message, SeNDORComm generates a packet and copies the immediate message into it. It piggybacks as many deferred messages as possible, the deferred messages being selected from the send queue in priority order. It then sends the packet comprising multiple messages down the stack. When an application requests SeNDORComm to send a deferred message, SeNDORComm stores the deferred message in the priority queue and returns immediately as long as the queue is not full and if no explicit packet is currently being sent. If the priority queue is full, SeNDORComm treats the deferred message as an immediate message. If an explicit packet send is in progress, SeNDORComm stores the deferred message in the queue but returns to the application only when the explicit packet send is done. This is to prevent the priority queue from overflowing by the application. Thus SeNDORComm rate-controls the application and partially prevents congestion from forming.

Each deferred message is assigned a threshold for staying in the queue based on its priority value. In cases when the deferred message has stayed for more than the threshold amount of time in the priority queue, an *explicit packet* is generated to send that message. This explicit packet piggybacks as many deferred messages as possible again in priority order. If sending the deferred message on the wireless link fails due to collision or lossy link, SeNDORComm stores the message back into its priority queue with priority value 1 (most urgent in the deferred message class). If transmission has been attempted more than the allowable number of times without success (three in our experiments), the message is dropped. The implication of this failure handling policy is discussed in Section 7.

SeNDORComm Receive. When a packet is received from GenericComm, SeNDORComm demarshalls the packet to retrieve the constituent messages in the packet and delivers one message at a time to the application. Since there can be multiple messages in a single packet, SeNDORComm uses a circular list, called receiver buffer that is provided by the application at initialization, to temporarily store the messages until they are consumed by the application. By delivering one message at a time, the receive message handler in the application need not be modified from that for GenericComm.

3.4. Application Interface and Implementation Details

The SeNDORComm interfaces are shown in Figure 1. In addition to the familiar send and receive interfaces, it provides a control interface SeNDORCommCtl that gives the flexibility for the application to turn on and off the generation of explicit packets. The SeNDORSend and SeNDORReceive interfaces are very similar to GenericComm Send and Receive interfaces. The send command in the SeNDORSend interface takes an urgency parameter, which is synonymous with the priority value for the message. The receive function in the SeNDORReceive interface has exactly the same syntax and semantics as the receive function in GenericComm's Receive interface except for the return value semantics. The return value of GenericComm receive command is a buffer where the next received message can be stored, while the return value of the SeNDORReceive function is indication of whether the passed buffer has been processed by the application or not. Due to space limitations, we refer the interested reader to [22] for complete implementation details. SeNDORComm has a low memory footprint of around 100 bytes and small code size of around 4 KB, when compiled using default settings in TinyOS development environment. Three configurations of buffers in SeNDORComm namely, minimum, medium (or typical) and large amount of buffers are shown in Table 1.

4. Analytical Evaluation

To evaluate SeNDORComm, we perform an analysis to derive *an upper bound* on the additional traffic injected into the network due to the explicit packets generated for the deferred messages The analytical result could be useful for guiding the choice of the deadline for deferred messages in a real deployment.

4.1. Assumptions and Notations

We make the following assumptions to make the analysis tractable. (1) A three-level hierarchy is assumed with sensing node, cluster head, and base station. (2) The clusters are all identical. Thus, analyzing a single cluster is sufficient. (3) The rate of immediate messages generated at a node is exponentially distributed with mean $1/\mu_i$ and for deferred messages the exponential distribution has mean $1/\Lambda$. (4) The priority of deferred messages is uniformly distributed in $[0, r]$. (5) In one packet only one deferred message can be piggybacked with an immediate message. Without this assumption, the queuing theory formulation would be overly complex since the service time would depend on the state of the queue.

Let n denote the number of nodes in the network, k the number of clusters, and m the number of nodes in each cluster ($= n/k$). Let f denote the compression factor at the cluster head i.e., the data size arriving at the cluster head divided by the data size sent by the cluster head to the base station. Let the rate of deferred messages with a given priority be $\lambda = \Lambda/(r+1)$. Let the transmission time of a packet in a CSMA network be exponentially distributed with rate μ_t.

4.2. Upper bound on the overhead traffic generated by individual nodes

The key observation for the analysis is that we can view the priority queue maintained by SeNDORcomm as an M/G/1 non-preemptive (head-of-the-line) priority queue with the immediate messages being considered as the server. A deferred message is *serviced* when an immediate message arrives and piggybacks the deferred message. So, a deferred message at the top of the queue can be considered to be under service until an immediate message arrives, piggybacks it, and gets sent out on the wireless channel. Thus, the service time (B) is the sum of inter-arrival time of immediate messages and the transmission time for the packet. Thus B follows a hypo-exponential distribution with parameters μ_i and μ_t.

To keep the analysis tractable, we do not take into account the "draining" effect of explicit packets on the priority queue. This is one of the factors pushing the final result to be an upper bound. The expected number of explicit messages generated is equal to the expected rate of messages arriving at the queue times the probability that a message waits more than the threshold wait time in the priority queue.

Let W_a be the random variable denoting the actual waiting time of a deferred message, W_p the waiting time for the deferred message in queue with priority p and B_{imm} the inter-arrival time of immediate messages. The relationship between them is shown in Eqs. (1),(2). Assuming B_{imm} and W_p are independent, we have Eq. (3). Let $E[N]$ denote the expected number of explicit messages generated by all queues in unit time and $E[N_p]$ the expected number generated in queue p per unit time. γ_p represents the threshold waiting time in queue p, $p = 1, \ldots, r$. Let γ represent the base threshold waiting time. $\gamma_p = \gamma + (p-1)/2$, $p = 1, \ldots, r$. We can write $E[N]$ and $E[N_p]$ as in Eqs. (4) and (5). To solve Eq. (5), we need the distribution of waiting time W_a, which requires the probability distribution of waiting time W_p for queue p. The priority queue with non-preemptive priority for *M/G/1* system has been well-studied. The first two moments of W_p are given in [16]. With B_i as the service time distribution of the queue, we have the first two moments of the waiting time as shown in Eqs.(6) and (7). B_i follows HYPO(μ_i, μ_t), for all p. Therefore, we can obtain the first, second, and third moments of B_i. Putting these in Eqs. (6) and (7), we obtain E[W_p] and $\sigma^2_{W_p}$ .The mean and variance of waiting time distribution can be used to obtain an upper bound for $P(W_p > \gamma_p)$ by using Chebyshev's inequality. For all $\varepsilon>0$, the inequality given in Eq. (9). Rewriting Chebyshev's inequality for $\gamma_p > E[W_a]$ (which is reasonable since the threshold should be longer than the average waiting time, otherwise the network will be overwhelmed with explicit packets), we have (using Eq.(5)) Eq. (10).

$$W_a = W_p + B_{imm} \tag{1}$$

$$E[W_a] = E[W_p] + (1/\mu_i) \tag{2}$$

$$\sigma^2_{W_a} = \sigma^2_{W_p} + (1/\mu_i^2) \tag{3}$$

$$E[N] = \sum_{i=1}^{r} E[N_p] \tag{4}$$

$$E[N_p] = \lambda P(W_a > \gamma_p) \tag{5}$$

$$E[W_p] = \frac{\sum_{i=1}^{r} \lambda_i E[B_i^2]}{2(1-\sigma_{p-1})(1-\sigma_p)} \tag{6}$$

$$E[W_p^2] = \frac{\sum_{i=1}^{r} \lambda_i E[B_i^3]}{3(1-\sigma_{p-1})^2(1-\sigma_p)} + \frac{\left(\sum_{i=1}^{r} \lambda_i E[B_i^2]\right)^2}{2(1-\sigma_{p-1})^2(1-\sigma_p)^2} + \frac{\left(\sum_{i=1}^{r} \lambda_i E[B_i^2]\right)\left(\sum_{i=1}^{r-1} \lambda_i E[B_i^2]\right)}{2(1-\sigma_{p-1})^3(1-\sigma_p)} \tag{7}$$

$$\text{where, } \sigma_p = \sum_{k=1}^{p} \rho_k = \sum_{k=1}^{p} \frac{\lambda}{\mu_s} = \frac{p\lambda}{\mu_s} \tag{8}$$

$$P(|W_a - E[W_a]| > \varepsilon) \le \frac{\sigma^2_{w_a}}{\varepsilon^2} \tag{9}$$

$$E[N_p] = \lambda P(W_a > \gamma_p) \le \lambda \frac{\sigma^2_{W_a}}{(\gamma_p - E[W_a])^2} \tag{10}$$

$$\zeta_{no\_debug} = m\mu_{data}(1 + 1/f) \tag{11}$$

$$\beta_{node} = \mu_{data} + \lambda + E[N] \tag{12}$$

$$\beta_{head} = \frac{(m * \mu_{data})}{f} + (m+1)\lambda + (m+1)E[N] \tag{13}$$

$$\begin{aligned}\zeta_{with\_debug} &= \beta_{head} + m\beta_{node} \\ &= \zeta_{no\_debug} + (2m+1)\lambda + (2m+1)E[N]\end{aligned} \tag{14}$$

$$\zeta_{overhead} = \frac{\zeta_{with\_debug}}{\zeta_{no\_debug}} = 1 + \frac{(2m+1)\lambda + (2m+1)E[N]}{\zeta_{no\_debug}} \tag{15}$$

4.3. Upper bound on relative network-wide overhead due to debugging

The traffic in the network is defined as the number of packets generated in the network per unit time. Let ζ_{no_debug} and ζ_{with_debug} denote the total traffic in a cluster respectively without and with SeNDORComm. The subscript "with debug" indicates that as an example the analysis takes all deferred messages are generated due to runtime monitoring (or debugging). For ζ_{no_debug}, the traffic in the cluster, includes the packets generated by m cluster nodes and the packets sent by the cluster head and is shown in Eq. (11). With SeNDORComm, at a cluster node the traffic is due to the immediate data messages, the highest priority messages arriving with rate λ, and the explicit packets. At the cluster head, we get an upper bound if every deferred message is forwarded to the base station instead of being consumed locally. This can occur say if the message is related to a detected error that needs to be forwarded to the base station for action. The traffic generated by a cluster node β_{node} and by a cluster head β_{head} is given in Eqs. (12) and (13). The total traffic generated in a cluster with run-time debugging (ζ_{with_debug}) is given in Eq. (14). Therefore, the upper bound on the normalized overhead generated due to run-time debugging ($\zeta_{overhead}$) is given in Eq. (15).

4.4. Results: Analysis and Discrete Event Simulation

Using the above analysis, we plotted the normalized overhead traffic in a cluster against the base threshold waiting time (γ). Given a γ, γ_p is defined as $\gamma + (p-1)/2$, p is the priority value. For the plot μ_{data} = 1 message/s. The maximum packet size in SeNDORComm is 65 bytes (58 bytes payload + 7 bytes header). For mica2 nodes running B-MAC, the size of preamble at 2.2% duty cycle is 1212 bytes. Therefore, the minimum transmission time of nodes running at 2.2% duty cycle is 0.511 seconds at 20kbps. This gives us μ_t = 1.96 messages/s. A plot is drawn for high (0.5 messages/s) deferred message arrival rates and is shown in Figure 2. The overhead incurred by the baseline approach (*aka* GenericComm) in which every deferred message is sent to the cluster head immediately can be calculated by using $E[N] = \Lambda$ in Eq. (15).

We also performed a discrete event simulation of the M/G/1 non-preemptive priority queue to validate that the analysis gives the upper bound. The simulation also cannot take into account the effect of an explicit message in draining the queues and therefore, it also gives an upper bound, albeit *a tighter one* than the analysis.

The result shows that for a high load scenario, SeNDORComm reduces the overhead of debugging by 26% even with a reasonably short baseline deadline of 10 seconds for explicit packet generation. The gains are less (about 7%) for a medium load scenario (0.1 deferred messages per second).

Table 1. Code and Memory Footprint of SeNDORComm integrated with LEACH

Components	ROM Size	RAM Size	Buffers Size
LEACH with GenericComm and Debugging	17884	811	0
LEACH with SeNDORComm and Debugging (1 buffer priority queue, 2 buffer receiver list)	21812	1118	138
LEACH with SeNDORComm and Debugging (5 buffer priority queue, 4 buffer receiver list)	21812	1351	426
LEACH with SenDORComm and Debugging (10 buffer priority queue, 4 buffer receiver list)	21812	1596	676

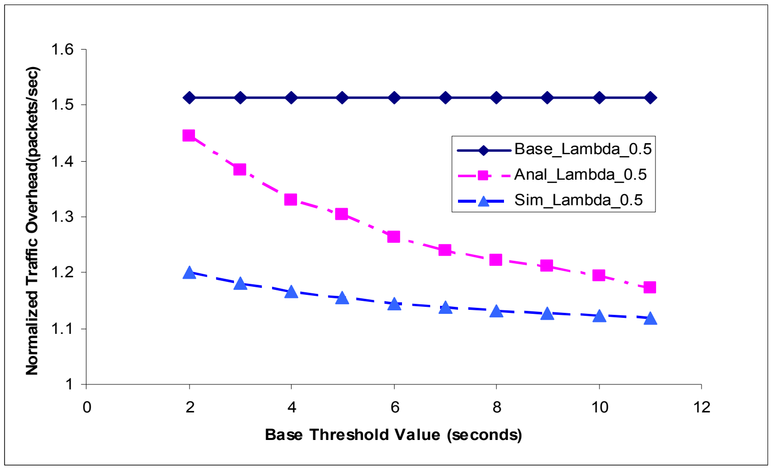


Figure 2. Normalized traffic overhead due to SeNDORComm and GenericComm (Base)

5. Experimental Evaluation

We designed a set of experiments to evaluate the main thesis of SeNDORComm that deferring lower priority messages and piggybacking them on immediate messages can improve energy efficiency and message reliability as well as its capability to handle bursty traffic. The first experiment is designed to measure the energy savings obtained using SeNDORComm. The second experiment is designed to evaluate SeNDORComm's performance under different load conditions in a real-world scenario. For this experiment, we used distributed debugging of a standard clustering protocol as a case-study.

In our experiments, we compare SeNDORComm against the baseline GenericComm, as state-of-the-art solutions (i.e., not GenericComm) use only one of our two principles (priorities and message pooling) and for a different purpose. For example, RAP uses priorities at the routing layer for scheduling packet transmission but does not do message pooling. AIDA does message pooling at the link layer but does not use priorities. Moreover, RAP and AIDA are not end-to-end and therefore, do not preserve the end-to-end nature of application-priority.

5.1. Energy Expenditure under Interference

In this experiment, we evaluate the energy savings of SeNDORComm compared to GenericComm in the presence of interfering traffic. We have shown in [22] that the fundamental requirement of SeNDORComm, namely the ability to pack multiple messages in a packet is met in sample indoor and outdoor settings without significant increase in loss rates.

Two Mica2 motes are kept at approximately 5 m distance and at 1 m height from the floor. The sender attempts to send 200 unique messages of which 25% are immediate and 75% are deferred. The sender retries a message three times before dropping it. For SeNDORComm, the sender retransmits only the immediate messages to have a fair comparison with GenericComm (since the SeNDORComm layer itself takes care of retransmitting the deferred messages thrice). The sender attempts to send a unique message every second if the radio is free, else, it waits for the next second. Therefore, the experiment period is the time taken to send 200 messages. The experiments are run in BMAC's LPL mode 3 (corresponding to 11.5% radio duty-cycle).

We perform three sets of experiments—with no interfering node, 3 interfering nodes and 5 interfering nodes. These emulate low, medium, and high contention networks respectively. In each set, the experiment is repeated 6 times for each of SeNDORComm and GenericComm, which gives acceptably low variance.

To measure the current used by the mote, the sender node was connected to the HP Agilient 3458A Multimeter over the span of entire experiment, which was 6 minutes. The current was sampled every 5 milliseconds. The energy spent by the mote is the product of current measurement, voltage (3 volts) and time (5 milliseconds). The total energy spent by the mote over the span of the experiment is the sum of the energy of all samples.

We have used the two performance metrics: (1) the total transmission energy spent per useful receive byte, where useful bytes are from messages that are not duplicates, (2) fraction of deferred and immediate messages received correctly.

In Figure 3 (a), we see that the energy required per useful byte received is considerably lower for SeNDORComm (43.5%, 44.2%, and 59.3% for low, medium, and high interference). This energy savings is due to piggybacking deferred messages on immediate messages, which reduces the fixed overhead cost associated with sending a packet. When the interference from other nodes increases, the energy spent increases for both communication layers due to the increased losses from collisions. However the increase is faster for GenericComm. By prioritizing and batching, SeNDORComm sends fewer packets in the network thereby reducing packet collisions and retransmissions.

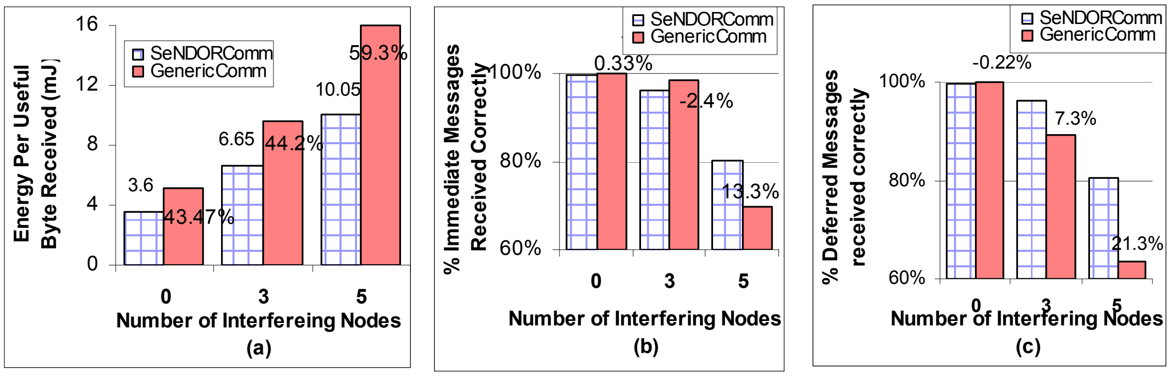


Figure 3. (a) Energy spent by the sender node per useful byte received for different levels of interference in network. (b) and (c)Fraction of immediate and deferred messages received successfully. The percentage numbers on GenericComm denotes the increase relative to the corresponding SeNDORComm case and in (a), the number on SeNDORComm denotes the absolute energy value.

Since SeNDORComm piggybacks deferred messages on immediate messages, it increases the possibility of immediate message getting dropped due to channel losses. However, we see from Figure 3 (b) that the percentage of immediate messages dropped is very low. The simple retransmission mechanism used by the application compensates for occasional losses. When interference increases, SeNDORComm achieves higher throughput with immediate messages than GenericComm as the packet losses due to collision starts dominating over channel losses. Finally, we notice that the fraction of deferred messages received by SeNDORComm is much higher than in GenericComm (Figure 3 (c)). Moreover, the fraction decreases much slower than in GenericComm with increasing amount of interference. This is due to the fact that deferred messages are piggybacked in SeNDORComm rather than each being sent as a separate packet. This causes less network contention and hence fewer losses.

5.2. Network Utilization under different loads

In this experiment, we study the network utilization under different load conditions in a real-world scenario and evaluate the ability of SeNDORComm to delay congestion collapse when the network is heavily loaded.

Checking invariants during run-time is an effective and widely used approach for debugging distributed systems [14][15]. In distributed debugging, invariants are checked at run-time. To check global invariants, debug messages are exchanged among nodes. An idea gaining ground in the WSN community is that distributed debugging is important for robust deployments of these networks [7][8][9]. However, the additional traffic introduced due to debugging can be significant under many different scenarios, such as, a change in the environment that results in multiple concurrent invariant violations and correlated failure of several sensor nodes. In these scenarios, it is important to detect and locate the error in the network promptly for a possible recovery. To achieve this, it is necessary to have the critical error information reach the base station. This is particularly difficult for the baseline communication layer to handle because the problem often manifests itself at the time when the available bandwidth is also constricted.

Thus, distributed debugging can create varying loads in the network and lends itself as a suitable case-study for this experiment. Therefore, we implemented distributed debugging of a standard clustering protocol called LEACH [5] using HSEND [7][8]. Since the interface of SeNDORComm to application is similar to that of GenericComm, the modifications required to convert our LEACH with HSEND code to use SeNDORComm were minimal. In LEACH, the nodes organize themselves into clusters, with one node in each cluster acting as the cluster head for one round. Each round is divided into election timeslots that are used to elect a cluster head, and data timeslots that are used to send data to the cluster head. In election timeslots, the self-elected cluster heads advertise their status. Nodes that are not cluster heads choose one of the cluster heads to join depending on received signal strength.

We created a 21 node network of Mica2 motes arranged in a 2x1 grid configuration with all nodes in the communication range of each other. In our experiment, the parameters used for LEACH are as follows. Each round has 27 equal timeslots, 20 for sending data messages and 7 time slots for cluster formation. Each slot is 2 seconds long. We used 2 clusters and 9 nodes join a cluster on average. Therefore, each node gets 2.2 timeslots per round. The cluster head has a compression factor of 3, i.e., for every 3 messages it receives, it sends one to the base station. We created a simple WSN application that sends one data message to the base station in its designated slot. Each data message is 14 bytes, debug message is 8 bytes, and the maximum payload length is 58 bytes for SeNDORComm. The application ran respectively on top of GenericComm and SeNDORComm.

H-SEND has an invariant that monitors the rate of successful transmission of sensed data (immediate messages) at each node. If the rate is below a certain threshold, it generates an error message with priority value 3 (a deferred message). We set the threshold to be slightly higher than the node's normal sensed data rate so that on average a debug message is generated at every check. We vary the frequency of checking the invariant to vary the load in the network.

A WSN application using LEACH can choose to store the debug messages generated at different points in time until its data slot or send the debug messages as and when generated. In the former case, the application implements a FIFO queue to store messages. Implementing such a queue is useful in low load scenarios where all the messages generated can be sent within the allotted slots in a round.However, there are several drawbacks to such a queue. Since it is a FIFO queue, important messages such as data messages may stay in the queue longer than it should. Due to limitations in the memory resources in motes and lack of piggybacking, the queue overflow is inevitable even under moderate loads. Since BMAC, the default link layer in TinyOS is a CSMA protocol, messages can be sent outside of LEACH data slots. Therefore, in our experiments, we made the WSN application send messages as and when generated for both GenericComm and SeNDORComm. Our results indicate that the reliability is not affected by sending messages out of schedule unless the network is heavily congested.

The metrics of interest are (1) *Goodput*, the rate of immediate messages that reaches the base station (2) *Transmission success ratio*, the ratio of the number of messages received by nodes in the network to the number of message sends attempted by nodes including retransmission. This indicates how efficiently the channel is used for communication. (3) *Reliability of immediate (deferred) messages*, the ratio of immediate (deferred) messages received by nodes successfully to the total number of immediate (deferred) messages sent by nodes.

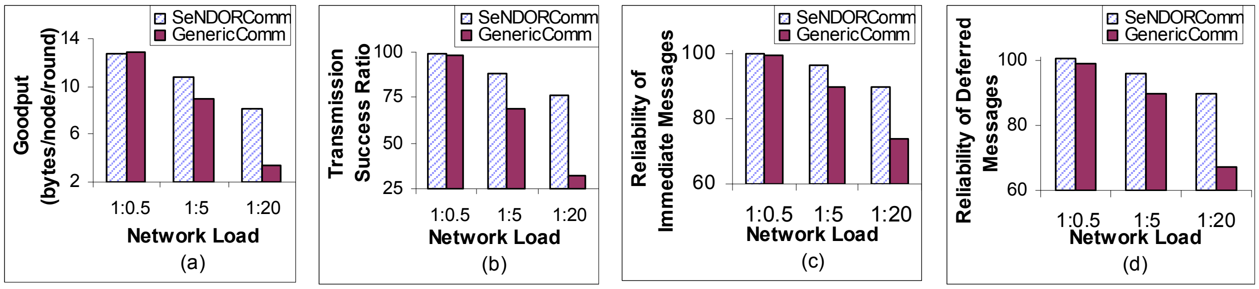


Figure 4. Behavior of SeNDORComm and GenericComm under varying load conditions on a 21 node test bed network

We ran all the experiments for 20 rounds. We observe the three output metrics mentioned above for SeNDORComm and GenericComm for varying load conditions. The variation in load is shown as the ratio of number of data messages (immediate messages in this case) to the number of debug messages (deferred messages in this case) generated. Figure 4 shows the behavior of SeNDORComm and GenericComm under light (1:0.5), medium(1:5) and heavy load(1:20) conditions. By piggybacking on immediate messages, SeNDORComm increases the likelihood of an immediate message being corrupted in the wireless channel. However, with a simple retransmission scheme, we see that for light load the reduction in goodput for SeNDORComm is low (-1.55%) and the reliability is almost the same for both (0.79% more than GenericComm). This corroborates our experiment 2 results in Section 6.2. Moreover, the goodput improves considerably as the load in the network increases (20.67% and 134.78% for medium and heavy loads). This is because the congestion is higher in GenericComm which affects the successful reception of immediate messages. In SeNDORComm, the batching of multiple messages into larger-sized packets alleviates the congestion to a certain extent. Likewise, the transmission success ratio of SeNDORComm improves (1.18%, 28.5%, and 131.99% for the light, medium and heavy loads) as it uses the network bandwidth more efficiently and therefore cuts down on the fruitless message sends that would collide and be lost on the congested wireless channel. Under heavy load, we see that the transmission success ratio for GenericComm has been reduced to 33% and this indicates congestion collapse as each message has to be sent four times.

In Figure 4(c), we see that with SeNDORComm the reliability improves as the load increases for both immediate messages (0.79%, 7.13%, and 21.20% for light, medium, and heavy loads) and deferred messages (1.53%, 7.24%, and 33.9% for the three loads). When the load is less, the packet losses are mainly due to channel losses and when the load increases, losses due to congestion start dominating. Hence, we see increased reliability benefit with SeNDORComm as the load increases.

6. Simulation Evaluation for Large Networks

To evaluate SeNDORComm for large WSNs, we used TOSSIM [21] to simulate experiment 2 for a 100 node MICA2 network. We used the same implementation of LEACH but with different parameters to scale it to 100 nodes, primarily an increase in the number of time slots allowed for sending the JOIN message and an application-level random back off mechanism to prevent collisions between multiple nodes sending in the same slot. Without the modifications, only 5% of the nodes were able to take part in data upload. We used LEACH with 5 clusters and 20 timeslots for sending data messages and 10 slots for cluster formation each slot being 10 seconds. To simulate wireless channel losses, we injected packet losses as observed in our experiment (8%).

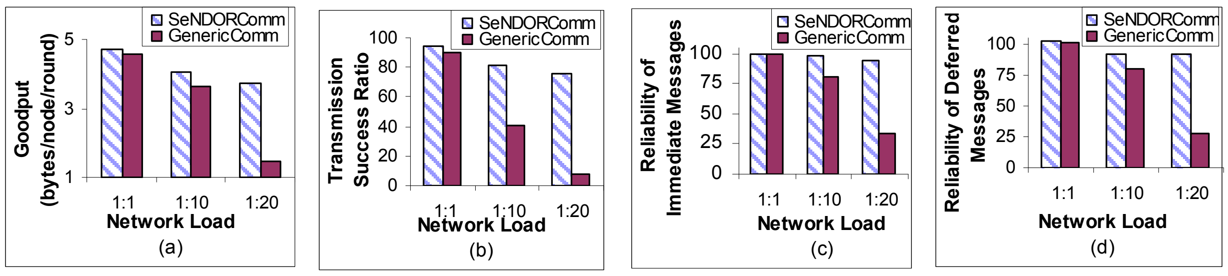


Figure 5. Behavior of SeNDORComm and GenericComm under varying load conditions on a 100 node network simulation.

Similar to experiment 3, we varied the network load by varying the number of debug message generated by HSEND. For each ratio, we ran the simulation for 20 rounds and averaged over the rounds for the results shown in Figure 5. The results from the test bed experiment and the simulation follow a similar trend. We see that SeNDORComm improves the goodput (3.29%, 12.09%, and 154.42% for light, medium, and heavy loads) and transmission success ratio (4.86%, 99.48%, and 830.98% for the loads) as the load increases. We see that under light load conditions the reliability of immediate messages in SeNDORComm is slightly less than that of GenericComm (-0.79%) owing to the channel losses. However, the reliability increases under heavier loads for both immediate messages (22.05% and 176.35%) and for deferred messages (13.68% and 231%) for medium and heavy loads. Under heavy load, we observe a congestion collapse in GenericComm, while this effect is not seen with SeNDORComm.

7. Discussion

Here we discuss some issues with the current design of SeNDORComm and methods to improve on them. First, it is conceivable that there are applications where any to any communication between any two nodes in the network is frequent. In such cases, SeNDORComm generates an explicit packet for the deferred messages if it does not expect an immediate message to the base station soon. SeNDORComm can improve energy efficiency even in these cases if the explicit packet contains multiple deferred messages.

Second, when SeNDORComm signals sendDone to the application for a deferred message, it takes over responsibility for sending the message out. However, if it fails a designated number of times, then the message is dropped and the application receives no notification. One may argue that if this is a critical message for the application, it should have been sent as a highest priority message in which case sendDone has the expected semantic that the message was transmitted successfully to the next node. Alternately, we can add a third phase to the split phase send operation whereby the node gets a later callback event with a success status when the message is sent successfully, or a failure status if it is not sent successfully.

Third, SeNDORComm provides a guarantee that a message send will be attempted by the threshold waiting time. The guarantee *does not* cover delivery or even a successful send attempt. The guarantee is a weak one for several practical reasons—the condition of the wireless channel cannot be predicted and the single timer used for aging messages in the queue has a fixed granularity. To improve matters, SeNDORComm could estimate the channel condition based on its transmission attempts and try sending a message in advance of the deadline based on an estimation of the lossiness of the channel.

Fourth, congestion can form even with SeNDORComm when a network is heavily loaded. The application can sense congestion with high packet loss and SeNDORComm's admission control can stop the application sending explicit messages for certain amount of time to alleviate congestion. Alternately, the congestion control protocols in [11][12] that are complementary to SeNDORComm can be used.

8. Conclusion

In this paper, we have presented the design and implementation of a communication layer called SeNDORComm that can handle messages with different priorities. It can buffer and piggyback messages which are not immediate so as to optimize the wireless channel usage. It respects latency bounds within which a message needs to be transmitted and it does not starve lower priority messages. Through experiments on a sensor network testbed, we show that packing multiple messages in a packet is possible without significant losses and the efficient use of the wireless channel results in lower energy consumption and increases the reliability of the end-to-end communication over the current default communication layer called GenericComm. In future work, we will be diagnosing problems in WSNs by correlating error messages. In addition, we are developing a compiler to automatically inject invariants in an application.

9. Acknowledgments

The authors would like to thank Mr. Jae-Woo Lee for helping with experiments and Mr.Douglas R. Herbert for providing source code of his LEACH implementation in Tiny OS.

10. References

[1] Lu, C., Blum, B. M., Abdelzaher, T. F., Stankovic, J. A., and He, T. RAP: A Real-Time Communication Architecture for Large-Scale Wireless Sensor Networks. In RTAS '02.

[2] He, T., Blum, B. M., Stankovic, J. A., and Abdelzaher, T. AIDA: Adaptive application-independent data aggregation in wireless sensor networks. In ACM Trans. on Embedded Computing Sys. pp. 426-457, May 2004.

[3] J. Polastre, J. Hui, P. L. J. Zhao, D. Culler, S. Shenker, and I. Stoica, "A Unifying Link Abstraction for Wireless Sensor Networks," SenSys 2005.

[4] J. Polastre, J. Hill, and D. Culler, "Versatile low power media access for wireless sensor networks," in Sensys, pp. 95-107, 2004.

[5] W. B. Heinzelman, A. P. Chandrakasan, and H. Balakrishnan, "An application-specific protocol architecture for wireless microsensor networks," IEEE Trans on Wireless Communications, vol. 1, pp. 660-670, 2002.

[6] T. He et al., "VigilNet: An integrated sensor network system for energy-efficient surveillance," ACM Transactions on Sensor Networks, pp. 1-38, February 2006.

[7] Herbert, D., Sundaram, V., Lu, Y., Bagchi, S., and Li, Z. Adaptive correctness monitoring for wireless sensor networks using hierarchical distributed run-time invariant checking. ACM Trans. on Autonomous Adaptive Systems, Sep. 2007.

[8] N. Ramanathan, K. Chang, R. Kapur, L. Girod, E. Kohler, and D. Estrin, "Sympathy for the sensor network debugger," Sensys 2005.

[9] D. Herbert, Y. H. Lu, S. Bagchi, and Z. Li, "Detection and Repair of Software Errors in Hierarchical Sensor Networks," SUTC, pp. 403-410, 2006.

[10] G. Tolle and D. Culler, "Design of an application-cooperative management system for wireless sensor networks," EWSN, pp.121-132, 2005.

[11] B. Hull, K. Jamieson, and H. Balakrishnan, "Mitigating congestion in wireless sensor networks," Sensys 2004 .

[12] C. Y. Wan, S. B. Eisenman, and A. T. Campbell, "CODA: congestion detection and avoidance in sensor networks," Sensys, pp. 266-279, 2003.

[13] Y. Sankarasubramaniam, I. F. Akyildiz, and S. W. McLaughlin, "Energy efficiency based packet size optimization in wireless sensor networks," in IEEE Workshop on Sensor Network Protocols and Applications (SNPA), pp. 1-8, 2003.

[14] M. Zulkernine and R. E. Seviora, "A Compositional Approach to Monitoring Distributed Systems," IEEE International Conference on Dependable Systems and Networks (DSN'02), pp. 763-772, 2002.

[15] G. Khanna, P. Varadharajan, and S. Bagchi, "Self Checking Network Protocols: A Monitor Based Approach.," SRDS, pp. 18-30, 2004.

[16] R. G. Miller, "Priority Queues," The Annals of Mathematical Statistics, vol. 31, pp. 86-103, 1960.

[17] Chatzigiannakis, I., Kinalis, A., and Nikoletseas, S. Priority based adaptive coordination of wireless sensors and actors. Q2SWinet '06.

[18] Ganti, R. K., Jayachandran, P., Luo, H., and Abdelzaher, T. F. Datalink streaming in wireless sensor networks. In SenSys, pp. 209-222, 2006.

[19] T. He, J.A Stankovic, C. Lu, T. Abdelzaher, "SPEED: A Stateless Protocol for Real-Time Communication in Sensor Networks," ICDCS, 2003.

[20] A. Arora et al., "A line in the sand: a wireless sensor network for target detection, classification, and tracking," Computer Networks, pp. 605-634, December 2004.

[21] Levis,P., Lee, N., Welsh, M., and Culler, D. 2003. TOSSIM: accurate and scalable simulation of entire TinyOS applications. ACM SenSys 2003.

[22] V. Sundaram, J.W. Lee, S. Bagchi, Y. H. Lu, and Z. Li .SeNDORComm: An Energy-Efficient Priority-Driven Communication Layer for Reliable Wireless Sensor Networks .Technical Report, School of Electrical and Computer Engineering. Purdue University. http://docs.lib.purdue.edu/ecetr/