Datalink Streaming in Wireless Sensor Networks

Raghu K. Ganti, Praveen Jayachandran*, Haiyun Luo, and Tarek F. Abdelzaher
Department of Computer Science
University of Illinois, Urbana-Champaign
email: {rganti2, pjayach2, haiyun, zaher} @uiuc.edu

Abstract

Datalink layer framing in wireless sensor networks usually faces a trade-off between large frame sizes for high channel bandwidth utilization and small frame sizes for effective error recovery. Given the high error rates of intermote communications, TinyOS opts in favor of small frame sizes at the cost of extremely low channel bandwidth utilization. In this paper, we describe Seda: a streaming datalink layer that resolves the above dilemma by decoupling framing from error recovery. Seda treats the packets from the upper layer as a continuous stream of bytes. It breaks the data stream into blocks, and retransmits erroneous blocks only (as opposed to the entire erroneous frame). Consequently, the frame-error-rate (FER), the main factor that bounds the frame size in the current design, becomes irrelevant to error recovery. A frame can therefore be sufficiently large in great favor of high utilization of the wireless channel bandwidth, without compromising the effectiveness of error recovery. Meanwhile, the size of each block is configured according to the error characteristics of the wireless channel to optimize the performance of error recovery. Seda has been implemented as a new datalink layer in the TinyOS, and evaluated through both simulations and experiments in a testbed of 48 MicaZ motes. Our results show that, by increasing the TinyOS frame size from the default 29 bytes to 100 bytes (limited by the buffer space at MicaZ firmware), Seda improves the throughput around 25% under typical wireless channel conditions. Seda also reduces the retransmission traffic volume by more than 50%, compared to a frame-based retransmission scheme. Our analysis also exposes that future sensor motes should be equipped with radios with more packet buffer space on the radio firmware to achieve optimal utilization of the channel capacity.

*First two authors equally contributed to this work

Sensys'06, November 1–3, 2006, Boulder, Colorado, USA.

Categories and Subject Descriptors

C.2.0 [**Computer Communication Networks**]: General—*data communications*; C.2.1 [**Computer Communication Networks**]: Network Architecture and Design—*wireless communication*

General Terms

Algorithms, performance, design, experimentation

Keywords

Throughput optimization, datalink layer, reliable communication, sensor networks, wireless networks

1 Introduction

Communications in wireless sensor networks as well as other asynchronous packet-switched wireless networks, usually involve a large per-frame overhead. It includes both the physical layer and MAC layer headers such as the physical layer preamble for per-frame synchronization, CRC for error detection, and unique sender/receiver identifiers for statistical multiplexing. As a result, the design for a reliable datalink layer often faces a dilemma when it comes to framing. For high utilization of the wireless channel bandwidth, larger frames are desired so that the constant per-frame overhead can be offset. However, large frames compromise the effectiveness of error recovery, where automatic retransmission request (ARQ) is usually applied. In an ARQ-based reliable datalink layer, the size of a frame is limited by the frame-error-rate (FER). Given a certain bit-error-rate (BER) that characterizes the wireless channel quality, FER ramps up quickly as the frame size increases. Since without error correction codes the entire frame has to be retransmitted even with a single error bit, the frame has to remain small.

The conflict between high channel utilization and effective error recovery seriously limits the achievable throughput even at the optimal frame size, particularly when the wireless channel quality is low and the BER is high. In fact, this is exactly what has been experienced in the framing at the TinyOS [8] datalink layer for wireless sensor motes [4]. Targeted at low-end, low-power embedded wireless transceivers of which the reliability is relatively low and BER is relatively high, a TinyOS frame is of 29 bytes by default, excluding the physical layer and MAC layer headers. Considering the 16-byte physical layer and MAC headers for MicaZ motes [4] equipped with 250Kbps CC2420 radio [1], the default framing at TinyOS datalink layer achieves less than 50% utilization of the wireless channel rate under typical channel BERs,

although the throughput is already near optimum[1]. The fundamental problem is that high throughput requires both high channel utilization and effective error recovery. Framing is unable to optimize these both at the same time.

One plausible solution to the above problem is to replace ARQ with sophisticated forward error correction (FEC) at the datalink layer. However, measurements [10, 2, 13, 6] have shown that the wireless channel quality changes quickly and unpredictably. It will be very difficult to gauge the FEC encoding redundancy to match the current channel error rate. Moreover, our measurements (see Section 4.1) on MicaZ motes and those [13] in 802.11 networks show that, even within a single frame the erroneous bits are highly clustered and bursty. Therefore, complex interleaving over large chunks of data is necessary in order for FEC to be effective. Recent work, SPaC [6] and MRD [13], have demonstrated the success in the application of packet combining [16, 3] and hybrid ARQ [12, 5] in asynchronous wireless networks. However, SPaC operates at the frame level, retransmitting encoded versions of the corrupted frames. MRD can be viewed as one type of spatial FEC in that it exploits the receptions of multiple erroneous versions of the same packet at multiple redundant 802.11 access points. MRD involves a computing complexity exponential in the number of blocks in a frame, and when the combination fails, it still retransmits the entire frame. Therefore, both SPaC and MRD share the same weakness with other datalink layer designs based on frame-level ARQ.

In this paper, we tackle the dilemma between large frame sizes for high channel utilization and small frame sizes for effective error recovery from another angle. Instead of further pursuing a better trade-off, we seek to *decouple* these two conflicting goals. We introduce blocks as the basic error recovery unit optimized for ARQ, and frames as the basic communication unit optimized for high channel utilization. To this end we propose Seda, a streaming datalink layer. Seda treats upper layer packets destined for the same next hop neighbor as a continuous stream of bytes. It divides the byte stream into blocks with simple per-block error detection. Erroneous blocks are identified and retransmitted, as opposed to retransmitting the entire erroneous frame. The size of a block is determined by the observed distribution of channel error bursts to optimize the effectiveness of ARQ. As a result, the size of the datalink layer frame is freed from the upper bound set by frame error rate (FER), which grows exponentially as the size of the frame increases. In fact, the frame size in Seda will only be constrained by other factors, such as the amount of buffer space available at the wireless transceiver firmware (e.g., 128 bytes in MicaZ's CC2420 radio) and the delay requirements of the network layer. Therefore, a frame can be substantially large [2] regardless of the channel quality, in great favor of high utilization of the wireless channel bandwidth without compromising the effectiveness of ARQ error recovery.

[1]This is shown in Figure 6. The details of the experiment and loss models are described in Section 4.

[2]Seda is targeted for applications that generate sufficient amount of data to necessitate throughput optimization.

Our contributions in this paper are three fold. First, we recognize the sub-optimality of the achievable throughput when framing is manipulated as the single tuning knob for both high channel utilization and effective error recovery. To the best of our knowledge, we design the first streaming datalink layer for fine-grained error recovery that is decoupled from framing. Second, our design is backed up with thorough analysis based on models derived from real wireless channel measurements. The analysis exposes how and to what extent Seda lifts the throughput upper bound, as compared with the bound achievable through framing alone. Finally, we implement Seda in the TinyOS programmable protocol stack, address many practical issues in efficient and robust protocol design, and evaluate the performance through experiments on a testbed of 48 MicaZ motes. Our experimental results show that Seda improves the throughput by about 25% under typical wireless channel quality, and reduces the percentage retransmission traffic by more than 50% compared to a frame-based retransmission scheme. Further, with capabilities for transmitting larger frames (more than 128 bytes), Seda will be able to improve throughput by upto 40% (as shown in Section 4).

The rest of the paper is organized as follows. We present the details of our protocol design in Section 2. Section 3 presents the implementation details of Seda. Section 4 provides an analysis based on models derived from real measurements of wireless channel errors. Section 5 provides a discussion of Seda. Section 6 describes the results of the evaluation through extensive simulations and experiments. We compare Seda with related work in Section 7. Finally Section 8 concludes this paper with directions for the future work.

2 Seda Design

For ease of exposition, we consider each data packet from the network layer as a single stream of B bit fixed-size logical blocks (we later specify what happens if the frame ends before the last block is full, and how the sender and the receiver can agree on a block size from time-to-time). We discuss how the network layer packet is split into blocks at the source and reassembled into the network layer packet at the receiver.

2.1 Overview

The basic Seda protocol works as follows. A packet from the higher (network) layer is divided into blocks, and each block is augmented with a 1-byte sequence number and a 1-byte CRC-8 for error detection. These blocks are then packaged into a frame and handed off to the MAC layer for transmission. When the receiver receives this frame, it identifies the blocks in error. After a certain number of frames are received, the receiver initiates the recovery, by sending a recovery frame in which the identifiers of erroneous or missing blocks are encoded. The sender responds to this recovery frame by retransmitting the blocks that the receiver requested. Since each block is self-identified with the sequence number, the retransmitted blocks can be packaged into the same frame with other new blocks. Therefore small frame transmissions are avoided. When all blocks of a packet

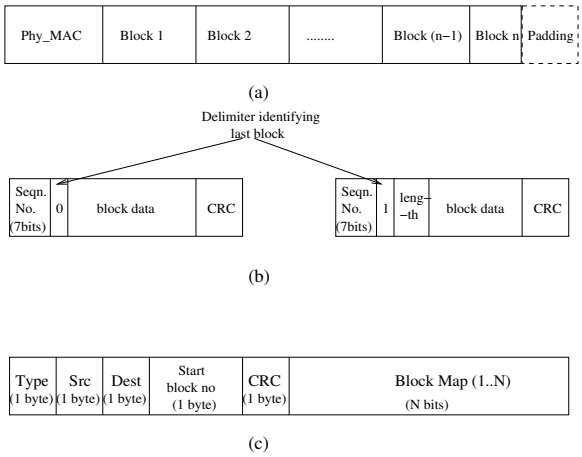

Figure 1. Seda message formats

are received correctly, the packet is forwarded to the network layer.

Augmenting a block with a 1-byte sequence number and a 1-byte CRC enables the receiver to identify the set of blocks that are in error. Since we consider data as a stream of bytes and not as a set of packets, the blocks in error can be retransmitted by the source regardless of their position in the byte stream. Block sequence numbers are assigned continuously, regardless of which packet they belong to. For example, if the first packet is split into blocks with sequence numbers 1 to n, then the second packet will be assigned block sequence numbers starting from $n+1$. The sender and receiver synchronize their initial sequence number during neighborhood discovery. The current sequence number is maintained as part of the neighbor states.

2.2 Protocol and Message Formats

The formats of a Seda data frame, a Seda block, and the Seda recovery frame are shown in Figures 1(a), (b), and (c) respectively. If the data packet size is not an exact multiple of the block size (B), then the last block is padded with zeros to make the block size constant. Note that a padded block is always the last block of a packet. We identify such blocks by one bit situated within the one byte sequence number. When this last block identifier is set, the byte next to the sequence number specifies the length of the data within this block. The block size remains the same, but the length of the data field within the last block is lesser by one byte. Note that if the packet data size is an exact multiple of the block data size, one block with zero data will have to be appended to mark the end of the packet.

The use of a one-bit last block identifier restricts the maximum block sequence number to 127. Since, within a single link blocks will be delivered in order, we reckon that this restriction will not be a serious limitation as long as recovery frames (serving as acknowledgments) are generated frequently enough. When the sending rate is high, the frequency of recovery frame generation should be correspondingly high so as to ensure that wrap-around conflict does not occur. For example, our analysis in Section 4 shows that typically a frame will contain less than 15 blocks. In this scenario as long as the sender stops and waits for recovery frame after every 4 data frame transmissions, the conflicts due to sequence number wrapping-around can be avoided.

At the receiver end, the block sequence numbers and the last block identifiers are used to re-assemble the blocks into network layer packets. When the number of corrupt (detected by CRC) or missing (detected by discontinuous sequence number) blocks is beyond a certain threshold (e.g., 16 in our implementation with 4-block frames), the receiver broadcasts a recovery frame. This recovery frame consists of a start block number, from where corrupted or missing blocks first occur, and a block-map. Each bit of the block-map indicates whether the corresponding block, following the block identified by the start block number in order, is received correctly or not. Thus, the recovery frame serves as the positive acknowledgment for blocks prior to the block identified by the start block number. Meanwhile it serves as negative acknowledgment for corrupted or missing blocks including and after the block identified by the start block number. Note that an entire frame can be missed when errors occur at the physical or MAC headers. Therefore, a receiver will send out the recovery frame after a certain timeout if there exists unacknowledged blocks, even if the number of corrupted and/or missing frames is below the threshold. A sender will retransmit blocks if no recovery message is received when its retransmission timer fires.

The format of the recovery frame is shown in Figure 1(c). The `Type` indicates a recovery frame, `Src` and `Dest` refer to the source and destination of the stream of data, respectively. The `Start block no` identifies the first corrupted or missing block. The `CRC` field contains the 1-byte CRC for all the above fields. The `Block Map` field is a bit-map (indicating whether the corresponding block has been received correctly by the receiver or not) for the set of consecutive blocks for which recovery has been initiated. In our implementation, the size of the block map is 16 bits. If the CRC check fails at the source, the recovery frame is dropped.

2.3 Seda Cooperative Streaming

Due to the broadcast nature of wireless communication, nodes close to the sender and the receiver are captured by the ongoing transmission. Cooperative communication has been explored in [6, 13]. We adopt this scheme as an optimization to Seda and claim no novelty in this regard. An intermediate helper node cooperates with the receiver by re-transmitting the corrupted blocks that the receiver has received. As observed in [18], the reception at the intermediate helper nodes will be superior to the reception at the receiver. Also, note that the block error probability at the receiver will be better when the helpers transmit (the correct blocks) as compared to when the source transmits. We use a simple learning automata based scheme to determine the helper(s) for each transmission. As cooperative communication is not the central focus of Seda, for brevity reasons, we do not discuss the details of how helpers are identified.

3 Seda Implementation

Seda is implemented as a link layer protocol which resides between the network layer and the MAC layer of the

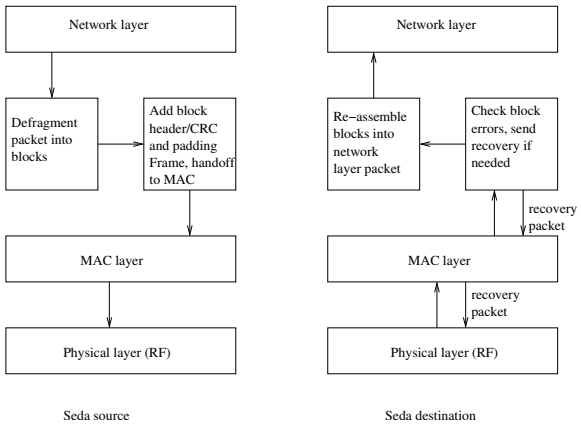

Figure 2. Seda in the communication stack

communication stack. Figure 2 presents the functions of Seda at both the sender and the receiver side. At the sender side, a packet from the network layer is divided into blocks and the block sequence number, CRC, and padding (if necessary) are added, as explained in Section 2. These blocks are packaged into a normal TinyOS frame and handed off to the MAC layer for transmission. On the receiver side, when a Seda packet is received the corrupted blocks are identified and a recovery frame is sent (after a certain number of frames have been received). The correctly received blocks are then re-assembled (based on the sequence numbers) and handed off to the network layer, as described in Section 2. The pseudo-code for the Seda sender and receiver are shown below.

Seda sender

```
Step 1 Divide network layer packet into blocks
Step 2 Add block sequence number and CRC, reframe
       and handoff packet to MAC layer for
       transmission
Step 3 If ACK is not received within a
       timeout_sender period
Step 4  Sender retransmits blocks
Step 5 else if ACK received
Step 6  Retransmit requested blocks (with new
        blocks if any) as determined from the
        BlockMap field of the ACK (recovery
        packet)
```

Seda receiver

```
Step 1 Identify corrupt/correct blocks
Step 2 If all blocks are correctly received
Step 3  Re-assemble blocks into a network layer
        packet and hand-off to network layer
Step 4 else if some blocks are corrupt or timer
       expires
Step 5  Buffer correct blocks
Step 6  Send ACK (recovery frame) after N frames
        are received or a period of timeout_receiver
        elapses, whichever is earlier
```

Seda has a code size of 3.2KB in ROM and occupies nearly 1.4KB in RAM. The majority of RAM consumption is the frame buffers (800 bytes), in order to buffer the blocks until all blocks of the corresponding packet are received, as discussed in Section 5.2.

Seda provides a simple interface with a `send` command and a `receive` event, where upper layers call the `send` with the appropriate parameters to transmit a packet and is signaled when a packet is received. Seda will work with any MAC layer, with the assumption that the MAC layer will not discard any packets with bit errors. Currently, our implementation uses the B-MAC MAC layer (which is the default MAC layer of TinyOS), and will work well with other MAC layers such as S-MAC [19], Z-MAC [15] or T-MAC [17].

Recently a unifying link abstraction for sensor networks (SP) has been proposed [14]. This abstraction resides between the datalink and network layers of the communication stack. SP provides shared neighbor management and a message pool for data transmission, whereas various MAC protocols determine sleep scheduling and access control to the wireless channel. Seda complements these protocols by providing a clean abstraction for error recovery. The concerns of increasing the throughput of the wireless channel are abstracted out into the datalink layer by Seda. Seda works well in conjunction with SP. Data transmission and reception are message oriented in SP, with a variable message length. SP provides Seda with the variable length message, which is divided into blocks and handed over to the MAC layer in frames (as explained in Section 2). When Seda receives all blocks of a SP message, it reassembles the message and hands it over to SP.

4 Model and Analysis

In this section, we justify many Seda design choices through modeling and analysis based on in-field measurements. We first present our measurements of the bit errors of the wireless channel between MicaZ motes with 250 Kbps CC2420 radios. Based on the measurements of the FER, FLR (frame-loss-rate), and block error rate, we derive a simple model that describes the bit errors of the wireless channel. We then derive the throughput increase achievable by decoupling framing and error recovery. Further, we analyze the block size that optimizes throughput. The analytical throughput gains and optimal block sizes are verified in Section 6 with experiments in real sensor networks.

4.1 Measurements and Model

We conducted a day-long indoor experiment in our lab to measure the wireless channel errors using the CC2420 radio on MicaZ motes. In our experiments, a sender continuously transmits four frames every second, each with a payload of 100 bytes. The destination was placed at a distance of 6 meters from the source. The typical FER was found to be 14.2%. Considering the data transmitted by the sender as a single byte stream, we compute the error rates of B-byte blocks, as shown in Figure 3. FER is independent of the block size, and hence it is a horizontal line in the figure. It is clear from the figure that block error rate increases almost linearly with increasing block size, and is always lower than the FER as expected. Hence, we perform error correction at

the block level rather than at the frame level.

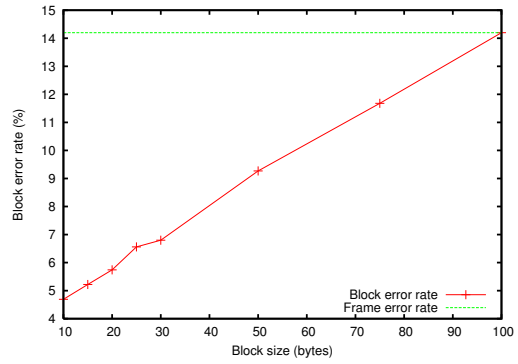

Figure 3. Block error rate vs. Block size (bytes)

For the same experiment, Figure 4 plots the bit error rate for each 200 bits (25 bytes) over a span of 100 frames. From the figure we can see that BER changes dramatically between zero and more than 50% during short intervals of 200 bit transmission time. It clearly demonstrates the burstiness of the wireless channel errors.

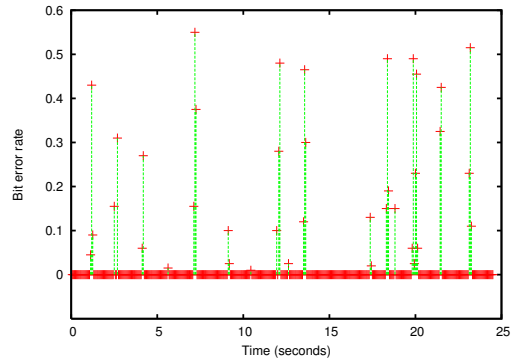

Figure 4. BER vs. time

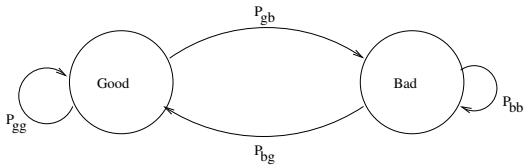

Figure 5. Gilbert-Elliot wireless channel model

To derive the wireless channel error model, we define *error cluster* as follows. An error cluster starts with a corrupted bit, and ends with another corrupted bit followed by 200 or more consecutive error-free bits. We define *inter-cluster* size to be the number of error-free bits between two consecutive error clusters. For the day-long experiment, we found that the mean and standard deviation of the size of error clusters are 250 and 160 bits, respectively. The mean and standard deviation of the inter-cluster size are 6600 and 6000 bits, respectively. Following the well-known two-state Markov chain Gilbert-Elliot (GE) model [7] as shown in Figure 5, the channel is in *Bad* state during error clusters, and in *Good* state otherwise. By this definition, BER in the good state is zero ($e_g = 0$). We further assume bit errors are uniformly distributed in the bad state with a BER, e_b. The state transition probabilities are P_{gg}, P_{gb}, P_{bb}, and P_{bg}, where P_{xy} denotes the probability of transition from state x to state y (g and b represent the good and bad state, respectively).

Loss Model	Mean Error Cluster Size (bits)	Mean Inter--cluster Size (bits)	e_b	Overall BER
1	250	1000	0.40	0.080
2	100	1000	0.40	0.036
3	386	3234	0.43	0.045
4	120	3234	0.36	0.013
5	386	9690	0.40	0.015

Table 1. Loss models

The high standard deviations of the sizes of error clusters and the inter-clusters (160 and 6000 bits, respectively as compared to their means 250 and 6600 bits, respectively) suggest a wide variation in channel error characteristics. Hence, choosing a single set of GE parameters to represent the wireless channel will be insufficient. Therefore, we evaluate Seda against five typical loss models we observed from our day-long experiments. Each loss model is defined by the mean error cluster size, the mean inter-cluster size, and the bit error rate in the bad state (e_b). Details of these loss models are shown in Table 1. Loss models 1 and 2 have the same mean inter-cluster size, but have different mean error cluster sizes. So do loss models 3 and 4. Loss model 1 represents scenarios of high error rate, and loss models 4 and 5 represent scenarios of low error rates. As we will see in the remainder of this section, the default TinyOS frame size (29 bytes excluding the physical and MAC layer overheads) actually leads to the optimal throughput with loss models 1 and 2.

$data$	Size of frame body data
phy_mac	Physical and MAC layer overhead
ack_oh	Per frame acknowledgment overhead
$block\_data$	Size of block data
$block\_oh$	Overhead of block
n	Number of blocks in a frame
P_g	Probability of being in good state
P_b	Probability of being in bad state
P_{bb}	Transition probability from the bad state to the bad state
P_{bg}	Transition probability from the bad state to the good state
P_{gb}	Transition probability from the good state to the bad state
P_{gg}	Transition probability from the good state to the good state
e_b	Bit error probability in bad state
FER	Frame error rate
FLR	Frame loss rate
KER	Block error rate

Table 2. Terms used in the analysis

For the following analysis, we define terms in Table 2. Given a loss model, we can obtain the probability of being

in the good state, $P_g = \frac{N_g}{(N_g+N_b)}$, where N_g and N_b are the mean sizes of error clusters and inter-clusters, respectively. The state transition probabilities can be derived as $P_{bg} = 1 - P_{bb} = \frac{1}{N_b}$ and $P_{gb} = 1 - P_{gg} = \frac{1}{N_g}$.

The probability that a frame is received correctly, (1 - FER), is shown as follows:

$$(1-FER) = P_g P_{gg}^{data+phy\_mac} + \sum_{i=0}^{data+phy\_mac-1} P_b P_{bb}^{i} (1-e_b)^i P_{bg} P_{gg}^{data+phy\_mac-i-1} + P_b P_{bb}^{data+phy\_mac} (1-e_b)^{data+phy\_mac} \quad (1)$$

Equation 1 consists of three terms. The first term indicates the probability of starting in the good state and remaining in the good state for the entire duration of the frame. The second term sums up the probabilities over all i, where the system starts in the bad state and remains in the bad state for i bits without incurring any errors and then transitions to the good state and stays there for the rest of the frame duration ($data + phy\_mac - i - 1$ bits). Finally, the third term specifies the probability of starting in the bad state and remaining in this state without any occurrence of errors. Note that, the second term is a geometric series, with a sum of $\frac{P_b P_{gg}(P_{gg}^{data+phy\_mac} - (P_{bb}(1-e_b))^{data+phy\_mac})}{P_{gg} - P_{bb}(1-e_b)}$. Reducing Equation 1, we obtain the following:

$$(1-FER) = P_{gg}^{data+phy\_mac}(\kappa + P_g) + (P_b - \kappa)(P_{bb}(1-e_b))^{data+phy\_mac} \quad (2)$$

where $\kappa = \frac{P_b P_{bg}}{P_{gg} - P_{bb}(1-e_b)}$.

The frame loss rate (FLR) is independent of $data$ but dependent only on phy_mac. Hence, it is given by:

$$(1-FLR) = P_{gg}^{phy\_mac}(\kappa + P_g) + (P_b - \kappa)(P_{bb}(1-e_b))^{phy\_mac} \quad (3)$$

Finally, the block error rate (KER) is obtained similarly as follows:

$$(1-KER) = P_{gg}^{block\_data+block\_oh}(\kappa + P_g) + (P_b - \kappa)(P_{bb}(1-e_b))^{block\_data+block\_oh} \quad (4)$$

From Equations 2, 3, and 4, we observe that under GE wireless channel error models the probability that a frame or block is received error-free decreases exponentially as the size of a frame or block increases.

4.2 Throughput Analysis

We now present an analysis of the optimal throughput that can be obtained using framing and using Seda, based on the FER, FLR, and KER derived in the above subsection. For datalink layer based on frame-level ARQ, it takes an expected $\frac{1}{1-FER}$ transmissions for the receiver to receive a frame correctly. Therefore, the expected throughput is:

$$throughput = (1-FER)\frac{data}{data + phy\_mac + ack\_oh}$$

Figure 6 plots the throughput for different values of $data$, for the five loss models. For this plot, we assume a phy_mac value of 16 bytes, which is the TinyOS physical and MAC layer overhead. We transmit an acknowledgment of size 23 bytes (including physical and MAC layer overheads), once every four frames. Hence the per-frame acknowledgment overhead (ack_oh) is about 6 bytes. From the figure, we observe that with TinyOS 29-byte frames, the achievable throughput varies from 31% to 52%. Furthermore, under optimal frame size the optimal throughput varies between 35% and 75%. Comparing the achievable throughput of loss model 2 and 4 (or loss model 3 and 5) we can see that with similar overall BERs, higher burstiness of the channel errors (represented by longer mean error busts and longer mean inter-cluster spacing) leads to higher throughput. Indeed, frame-level ARQ has the worst performance if the bit errors are distributed uniformly (not shown in Figure 6) or when the channel errors are the least bursty.

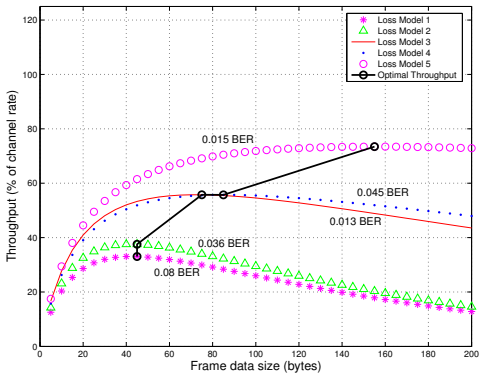

Figure 6. Throughput vs. Frame size

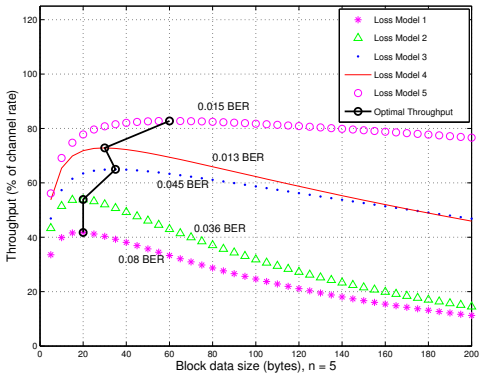

Figure 7. Throughput vs. Block size (5 blocks per frame)

With Seda block-level ARQ applied, the block error rate (KER) can be derived given the block size ($block$), the sum of the data ($block\_data$) and per-block overhead ($block\_oh$). It takes an expected $\frac{1}{1-KER}$ number of transmissions to deliver one block. Note that when an error occurs at the physical or MAC header, the receiver either misses the entire frame (i.e., when errors occur at the physical layer preamble), or has to drop the frame since it cannot be sure of the real sender and the intended receiver of the frame. Assuming that a frame carries n blocks, the achievable throughput is:

$$throughput = \frac{(1-FLR)(1-KER) \times n \times block\_data}{n \times (block\_data + block\_oh) + phy\_mac + ack\_oh}$$

Assuming a 2-byte per-block overhead ($block\_oh = 16\ bits$, 1-byte sequence number and the 1-byte CRC as elaborated in Section 2.2), 5-block frame ($n = 5$), we plot the throughput for different block sizes in Figure 7 with the CC2420 radio parameters and Equation 2 and 4 plugged in. As compared to the frame based retransmission scheme (Figure 6), the optimal achievable throughput increases by 8-20% with different loss models. Similar to frame-level ARQ, the throughput increases with increasing mean inter-cluster space even with the same overall BERs.

Several interesting observations can be drawn from Figures 6 and 7. First, the block size that optimizes throughput increases with increasing mean inter-cluster size. This can be explained as follows. Observe that the error clusters are small enough to be contained within a block or two. Since, the mean inter-cluster size is much larger than the mean error cluster size, only one in several blocks (say m) will be corrupted on an average. Also, as the mean inter-cluster size increases, m increases. Hence, when the mean inter-cluster size between clusters is larger, the reduced overhead of using larger block sizes for the m blocks offsets the overhead of retransmitting one larger corrupted block. This results in a larger block size that optimizes throughput. The second important observation is that the block size that optimizes throughput is nearly independent of the size of the error cluster (loss models 1 and 2, and loss models 3 and 4 have different mean error cluster sizes, but have the same mean inter-cluster size). The reason is that the variation in the inter-cluster size causes a significant change in the number of error clusters in any given time period (and hence a significant change in the overall BER). The variation in the error cluster size changes the number of bits in error within each cluster and not the inter-cluster size (and the size of the error cluster is an order of magnitude lower than the size of the inter-cluster), and hence does not have as significant an impact on the BER and throughput. This observation sheds some light on block size adaptation as we will discuss in Section 5. Finally, we observe that a block size of 20-25 bytes is nearly optimal for most channel conditions, except for loss model 5 where the BER is low and channel errors are the most bursty. We evaluate Seda with block sizes of 20 and 25 bytes in our simulations and experiments.

We show the impact of Seda frame size on achievable throughput in Figure 8. From the figure we can clearly see that the throughput increases *monotonically* as the frame size increases. This is clearly different from the datalink layer design based on frame-level ARQ, where large frame sizes lead to lower throughput due to the degraded performance of error recovery. With Seda, the size of a frame is freed from the constraint of effective error recovery. In our implementation of Seda on MicaZ motes, the frame size is actually bounded by the available packet buffer (i.e., 128 bytes) at the CC2420 radio firmware.

We finally show the impact of Seda frame size on the throughput gain against the throughput of frame-level ARQ

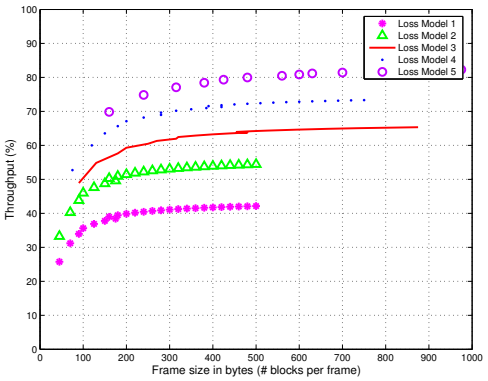

Figure 8. Throughput of Seda vs. Frame size (number of blocks per frame)

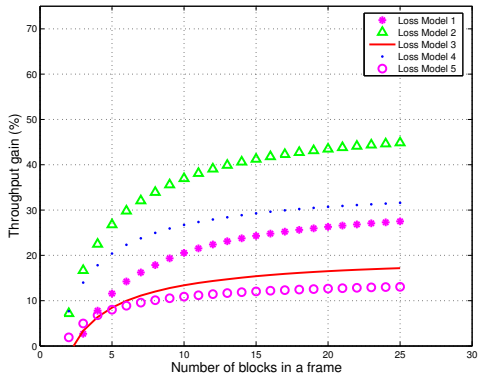

Figure 9. Throughput gain of Seda vs. Number of blocks per frame

in Figure 9. We observe that the throughput gain is larger as we increase n, which relates to larger frame sizes. For example, when $n = 20$ the optimal achievable throughput gains vary between 15% and 45%. Another interesting observation is that the throughput gain increases with decreasing mean error cluster size. This can be explained as follows. When the mean error cluster size is smaller, the block error rate is lower, while the frame error rate does not change due to the fact that most error clusters will be contained within a single frame (frame size is more than 800 bits, while the error cluster sizes are between 100-400 bits). Also, recall that the throughput of frame-based retransmission is dependent on the frame error rate, while the throughput of Seda is dependent on the block error rate. Hence, the throughput gain of Seda will be higher with smaller error clusters. Finally, *our choice of 4 blocks per frame in our implementation with MicaZ motes (Section 3), as a result of the 128-byte packet buffer space at CC2420 radio firmware, does not fully exhibit the power of Seda*. We expect future sensor mote radios to be equipped with larger packet buffer (around 200∼300 bytes), so that the system can benefit from the full strength of Seda.

5 Discussion

In this section, we discuss the block size adaptation and the frequency of recovery frame transmission.

5.1 Block size adaptation

In the design of Seda, we have assumed that network layer packets are split into fixed size blocks. However, the block size that optimizes throughput could vary with the channel error characteristics. From our analysis and experimental studies, we observe that although the block size that optimizes throughput varies with different error characteristics, a block size of 20-25 bytes provides near-optimum throughput under a majority of channel conditions. Only under high loss rates is the throughput of Seda with a block size of 25 bytes significantly lower compared to the throughput when the block size is 20 bytes. This can be attributed to the sharp rise and shallow drop-off in the throughput with increasing block size, as shown in Figure 7. In favor of lower complexity, we do not incorporate block size adaptation in our current implementation.

If block size adaptation is desired, a simple out-of-band signaling mechanism can be employed by the source and the destination to agree on a block size from time-to-time, based on the measurements of error characteristics of the wireless channel at that time. By comparing the corrupted block with the correct block received after retransmission, the receiver can measure the size of the error clusters, the interspacing between error clusters, and the bit error rate within error clusters (e_b), from time-to-time. These values can be plugged into the model described in Section 4, to compute the block size that optimizes throughput. If this block size is significantly different from the block size currently being used, the receiver could use out-of-band signaling to inform the source to transmit using the new block size. Note that, this mechanism requires the receiver to buffer the corrupted block until the block is received correctly. If buffer constraints prevent storing the corrupted blocks, the receiver could simply calculate the average number of consecutive blocks received correctly (over a certain duration) as a measure of the interspacing length between error clusters. From our analysis in Section 4, we observe that the error cluster size and e_b do not significantly affect the block size that optimizes throughput. So, the receiver could use default values for the error cluster size and e_b, and use the model to determine the block size that optimizes throughput.

5.2 Frequency of recovery frames

The frequency at which recovery frames are sent is determined by two factors: the delay which the network layer and the application can tolerate, and the available datalink layer buffer space. For a packet to be handed off to the network layer, all the blocks of that packet and of prior packets need to be received correctly. Thus, the delay incurred by the network layer is directly affected by the frequency of recovery frame transmission. Furthermore, when a frame is received, the blocks that are in error are discarded and those blocks which have been received correctly are buffered. If the frequency of recovery frame transmission is small, then the number of blocks that are buffered is high, demanding for more buffer space at the datalink layer. In our implementation, we use a buffer size of 800 bytes (for 8 100-byte frames). We send a recovery frame every 4 frames. We observed from our experiments that the recovery is always completed before the buffer overflows. The frequency of sending recovery frames can be adapted based on the observed error rate to control the latency involved in error recovery.

6 Performance Evaluation

In this section, we evaluate the performance of Seda. In Section 6.1, we conduct extensive simulation experiments and study the throughput and percentage retransmission traffic for Seda and compare the same with a Frame based ARQ (FARQ) scheme and with cooperative-Seda. We then provide results from an implementation of Seda on a testbed of 48 MicaZ motes in Section 6.2.

6.1 Simulation Experiments

In order to evaluate the performance of our proposed recovery mechanism, we conducted simulation experiments on TOSSIM [11], a TinyOS simulator. We conducted extensive experiments on a 2x2 and 3x3 grid network. For the 2x2 grid network, nodes on the main diagonal were chosen as the source and the destination. For the 3x3 grid, we vary the distance of the destination from the source. We compare the performance of Seda with FARQ. We implemented FARQ in the data-link layer, above the MAC layer, similar to the implementation of Seda (by default, the current implementation of TinyOS, does not perform any MAC-layer recovery). To make a fair comparison, the destination attempts to recover each block or packet only once. We used TOSSIM's in-built lossy model [11], which is based on empirical measurements of link losses for different distances between the sender and the receiver. The lossy model models interference and corruption, but it does not model noise. Also, TOSSIM does not model the bursty nature of errors in the wireless channel. The model is based on uniformly generating bit errors given the bit error probability on each link. However, we observed a large mean and standard deviation of the interspacing between observed bit errors, similar to the observations from the experiment on MicaZ motes described in Section 4. For instance, for a distance of $10\sqrt{2}$ meters between the source and the destination, the mean interspacing length between errors was around 4000 bits, while the standard deviation was 3200 bits. Since from our analysis we observe that the block size that optimizes throughput is mostly dependent on the interspacing between error clusters, and not so much on the mean error cluster size (as long as the mean error cluster size is an order of magnitude smaller than the mean interspacing length), we believe that TOSSIM's loss model is still reasonable to test Seda. TOSSIM's simulation environment permits us to test Seda for a large set of nodes and simultaneous flows, which would be extremely difficult in a real testbed of motes.

The default simulation parameters that we chose include a grid spacing of 10 meters, packet data length of 100 bytes (without considering any overheads), and a block data size of 25 bytes. The physical and MAC layer overheads per packet for TinyOS is 16 bytes. The per-block overhead of Seda is 2 bytes. Thus for Seda, the frame consists of 100 bytes of payload in four blocks, overhead of 8 bytes for the blocks, and 16 bytes physical and MAC layer overheads, resulting

in a frame size of 124 bytes. This is close to the maximum frame size of 128 bytes attainable using TinyOS, due to the hardware limitation of the CC2420 transceiver buffer size. For FARQ, we consider two frame data sizes of 29 bytes and 100 bytes. While 29 bytes is the default frame data size of TinyOS, 100 bytes is close to the maximum achievable frame data size. In addition to the data payload of 29 or 100 bytes, the physical and MAC layer overheads account for 16 bytes, resulting in a frame size of 45 or 116 bytes, respectively. In the rest of this section, when we refer to FARQ without specifying the frame size, we refer to a frame size of 116 bytes. In all figures that concern averages, we have presented standard deviations over 30 simulation runs, with each run consisting of 400 network layer packets.

Figure 10 shows the throughput obtained by Seda when compared with FARQ for different block sizes (FARQ does not use the block sizes). The destination was placed at a distance of $10m$ from the source. We observe that the throughput initially increases with block size, and then decreases beyond a block data size of 25 bytes. At the optimal block size, we notice that Seda improves the achievable throughput by nearly 25% compared to FARQ. We observe that with increasing block size, the average throughput for Seda initially increases and then decreases. This is similar to the observation in Section 4.

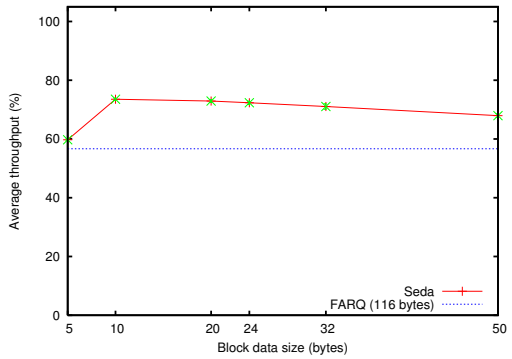

Figure 10. Throughput comparison of Seda and FARQ scheme for different block sizes

To study the performance of Seda under different channel characteristics, we varied the distance between the source and the destination. We considered a grid of nine nodes. The source was fixed at node 0. Five destination positions were used, which are nodes 3, 4, 6, 7, and 8. This corresponds to distances of 10, $10\sqrt{2}$, 20, $10\sqrt{5}$, and $20\sqrt{2}$ meters, respectively. Figure 11 shows the throughput achieved by FARQ for two different frame sizes of 45 and 116 bytes, Seda, and cooperative-Seda. For low loss rates (short distances) the overhead of using a small frame size for FARQ, results in poor throughput compared to FARQ using a larger frame size. However for higher loss rates, transmitting smaller frames results in more packets being delivered and hence a higher throughput. The performance gain achieved by Seda as compared to the FARQ scheme is the ratio of the throughput improvement obtained by Seda compared to the throughput of FARQ, expressed as a percentage. This is plotted in Figure 12. At very low loss rates, the throughput of Seda and FARQ are comparable. At medium and high loss rates, the throughput increase that Seda achieves as compared to FARQ varies from 15% to as high as 100%. We also observe that the performance gain is higher for larger distances. Cooperative-Seda further helps increase throughput by 5% to 10% compared to Seda, at high loss rates. This is because, the reception at the intermediate helper nodes is superior compared to the reception at the receiver. Also, the block error probability will be lower at the receiver when the helpers retransmit as compared to when the source retransmits.

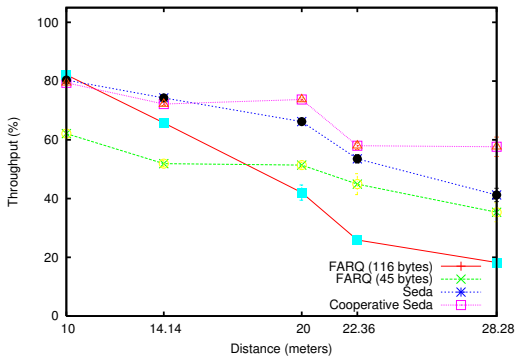

Figure 11. Throughput comparison of Seda and FARQ for different distances betn. the src and dest

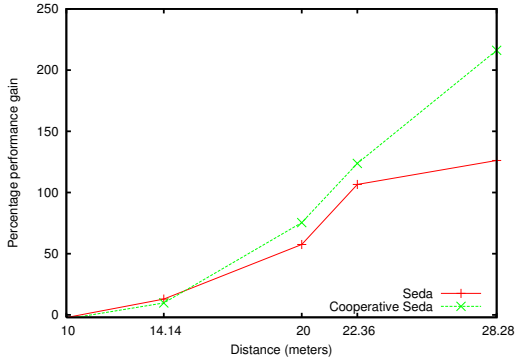

Figure 12. Performance gain of Seda and coop. Seda compared to FARQ for different distances betn. the src and dest

The percentage retransmission traffic with varying distances for FARQ, Seda, and cooperative-Seda is plotted in Figure 13. Seda and cooperative-Seda reduce the percentage retransmission traffic by about 40% when compared to FARQ.

To study the performance of Seda for different block sizes and error rates, we measured the average throughput and percentage retransmission traffic for three different distances between the source and the destination (namely $10\sqrt{2}$, 20, $10\sqrt{5}$ m), and for four different block sizes (namely 10, 20, 25, and 32 bytes). Figure 14 plots the average throughput. Similar to the trend observed in the analysis, the throughput initially increases with block size and then decreases. We notice that the optimal block size is around 20 bytes for distances of $10\sqrt{2}$ and 20m, while it is close to 10 bytes for a distance of $10\sqrt{5}$m. This is in accordance with the intuition

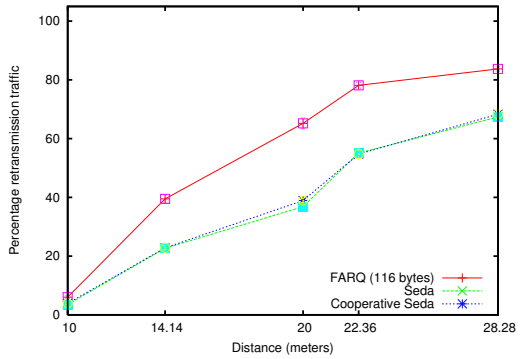

Figure 13. Percentage retransmission traffic of Seda and FARQ for different distances betn. the src and dest

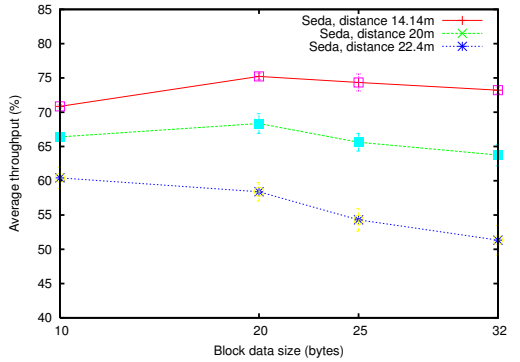

Figure 14. Average percentage throughput, when block size is varied for different distances

that, when loss rates are higher (distance between source and destination is more), the interspacing lengths between errors are lower, and the overhead of using lower block sizes is offset by the reduction in the number of retransmitted bits.

Figure 15 plots the percentage retransmission traffic for different distances and block sizes. We observe that, as the block size increases, the percentage retransmission traffic for Seda also increases. This is because, the source transmits at the granularity of blocks, and for larger block sizes, the block would consist of bytes that the receiver has already received correctly. For FARQ, the granularity is a packet, which will contain more redundancies. Hence, Seda has a lower percentage retransmission traffic compared to FARQ.

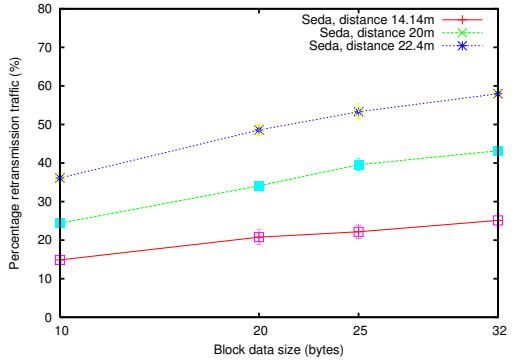

Figure 15. Percentage retransmission traffic, when block size is varied for different distances

Table 3 compares the block sizes that optimize throughput for different distances, obtained from simulations with those obtained from analysis. For each of the distances considered, we measured the mean interspacing length between errors. As the mean error cluster size and the bit error rate within error clusters (e_b) do not significantly affect the burst size that optimizes throughput, we use default values of 250 bits for the mean error cluster size and 0.4 for e_b. This is the approximation mechanism proposed in Section 2 for the receiver to measure the block size that optimizes throughput without having to buffer the corrupted blocks. The measured value of the mean interspacing length between errors and the default values of the mean error cluster size and e_b are plugged into the model proposed in Section 4, to obtain the block size that optimizes throughput based on our analysis. From the table, we notice that the block sizes that optimize throughput obtained from simulation and analysis for different error rates are comparable.

Dist. betn. src and dest.	Optimal block size (sim.)	Optimal block size (analysis)
14.14	20	23
20	20	20
22.36	10	15

Table 3. Comparison of the optimal block size obtained from simulations and analysis for different distances

To study the performance of Seda in a large topology with several simultaneous flows, we considered a 15×15 grid of nodes, with a grid spacing of 8 m. The communication range of each mote was about 32m. In this topology, we studied the throughput of ten simultaneous Seda flows. The source of each flow was chosen uniformly at random. For each source, the destination of the flow was chosen at random from amongst the set of nodes within the communication range of the source. Each source transmitted two packets every second. Figure 16, shows the grid topology and the flows numbered from 1 to 10. The arrow head on each flow points in the direction of the flow (from the source to the destination). Figure 17 shows the throughput for each flow, averaged over 20 runs, each of 75 seconds. From Figure 16, we observe that the ten flows form four clusters, where each cluster represents a set of mutually interfering flows. Flows 1, 3, and 4 form one such cluster, flows 7 and 10 form another, flows 2 and 6 form the third, and flows 5, 8, and 9 form the fourth. We observe that the throughput of flows 2 and 6, and that of flows 7 and 10 are higher than those of the other flows. This can be attributed to the fact that clusters formed by these flows have only two flows interfering with one another (while the others have three interfering flows), resulting in reduced error. Further, the destinations of flows 2 and 6 are far-removed from one another and hence, each of them have only a reduced interference from the other flow. This explains the higher throughput of these flows compared to flows 7 and 10, whose destinations are close to each other.

6.2 Experiments on MicaZ Motes

We evaluated the performance of our scheme on an indoor testbed of 48 MicaZ motes to establish the effectiveness of

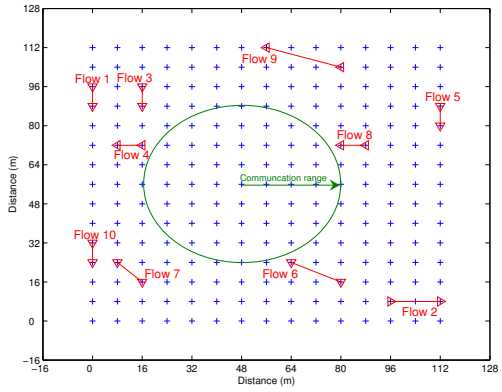

Figure 16. Grid topology with 10 simultaneous flows

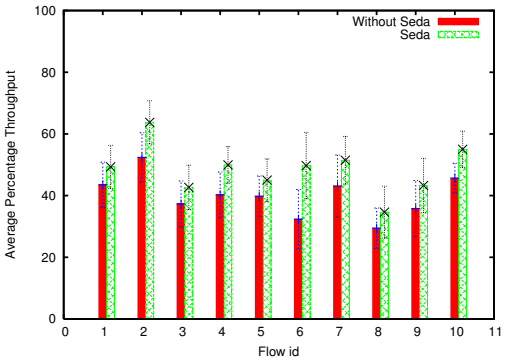

Figure 17. Average throughput of each of 10 simultaneous Seda flows

our scheme in reality. MicaZ motes are matchbox-sized microcontroller devices with wireless communication capabilities [4]. The MicaZ motes have a Chipcon-CC2420 radio, with a radio frequency of 2.4 GHz and a maximum data rate of 250 Kbps. The CC2420 radio includes a digital direct sequence spread spectrum modem. The testbed is arranged as a 8×6 grid, with an interspacing of 0.6 meters. The motes transmit at a power of $-18dBm$. The sender and the receiver are placed at the corners of the diagonal. Unless otherwise specified, the motes other than the source and the destination do not communicate with each other.

Default values used in our experiments are as follows. The sender(s) transmits two packets every second, each consisting of four blocks. Each block's data size is 25 bytes and has a two byte overhead. Hence, the total size of the packet without physical and MAC layer overheads is 108 bytes. We consider two frame data sizes for FARQ (without physical and MAC layer overheads), namely 29 and 100 bytes. For FARQ, the sender transmits two packets every second, each of size 100 bytes, excluding the physical and MAC layer overheads.

Figure 18 plots the throughput of Seda and FARQ over a duration of half an hour. Each point in the graph is obtained by averaging over 80 packets. We observe that, FARQ experiences high fluctuations in the throughput. On the other hand, Seda achieves a nearly consistent throughput.

For five different block sizes, we measured the throughput

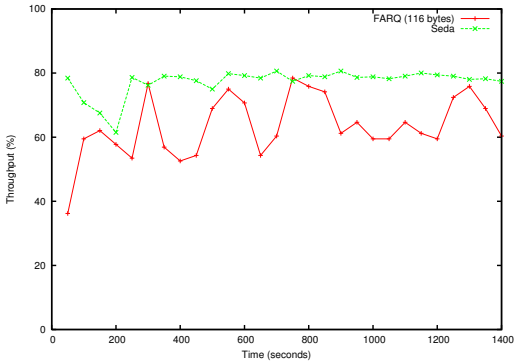

Figure 18. Throughput vs. Time for Seda and FARQ

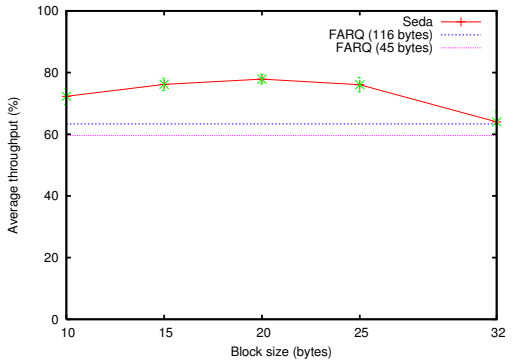

Figure 19. Throughput vs. Block size for Seda and FARQ

of Seda and FARQ , which is plotted in Figure 19. Each point in the graph represents the observed throughput over a half hour duration. For FARQ, we considered two frame data sizes, 100 bytes and 29 bytes. Frame data size of 100 bytes has a low per-frame overhead, but high loss rate, while a frame size of 29 bytes has a high per-frame overhead, and a low loss rate. By decoupling these two factors Seda is able to achieve better throughput than FARQ with either frame size. We notice that the optimal throughput for Seda is achieved at a block data size of about $20-25$ bytes. This figure is in accordance with our analytical model presented in Section 4. The performance gain achieved by Seda as compared to the FARQ scheme with a frame data size of 100 bytes, is the ratio of throughput improvement obtained by Seda compared to the throughput of FARQ, expressed as a percentage. Table 4 shows the percentage performance gain achieved by Seda. The throughput increase achieved by Seda when compared to FARQ is about 20% at the optimal block size.

To study the effect of one Seda flow on another, we measured the throughput of two simultaneous Seda flows for a period of 2000 seconds. The sources of both these flows send two packets every second. In this experiment, the source and destination of flow 1 were placed at the ends of the main diagonal, and that of flow 2 were placed at the ends of the cross diagonal. The observed throughput for the two flows is plotted in Figure 20. We define performance gain in throughput as the ratio of the throughput improvement of Seda compared to that of the baseline throughput without retransmissions, expressed as a percentage. This performance gain of Seda is plotted in Figure 21. We observe that, Seda flow 2

Block size	Performance gain
10	14.09
20	22.90
25	20.03
32	1.02

Table 4. Percentage performance gain of Seda for different block sizes

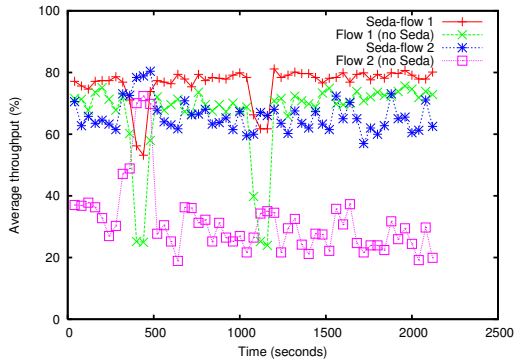

Figure 20. Throughput vs. Time for two Seda flows

experienced a poorer channel quality as compared to flow 1. Nevertheless, at most time instances, the throughput improvement obtained using Seda for flow 2 (compared to the throughput without any recovery mechanism) was more than 100%, which is much higher than the performance gain for Seda flow 1. This shows that for higher loss rates (as experienced by Seda flow 2), the performance improvement is higher. Notice the wide variation in the performance gain achieved by Seda from Figure 21. This large variation is explained by the bursty nature of the wireless channel.

We studied the performance of Seda under increasing external interference load conditions. We measured the throughput achieved by FARQ and Seda (for block data sizes of 20 and 25 bytes) at external load levels of 4, 8, 12, and 24 times the base Seda flow (2 packets every second), which is plotted in Figure 22. For 4 and 8 times the base Seda flow load, we induced load by introducing nodes that send packets periodically at the rate of four packets per second. For loads of 12 and 24 times the base Seda load, we used 1 and 2 nodes, respectively, transmitting packets using a split-phase send approach. Here, the interfering nodes continuously send one packet after another without any pause, which corresponds to about 24 packets every second or 12 times the base Seda load. We observe that Seda throughput is nearly unaffected when the external load is low. Under higher load conditions, the throughput degradation is graceful. Table 5 shows the performance gain in throughput achieved using Seda with a block size of 20 bytes, compared to the throughput of FARQ, for the different external interference loads considered. We observe that Seda achieves a performance gain of up to 25% compared to FARQ. Further, we observe that the throughput of Seda with block sizes of 20 and 25 are nearly equal for nominal load conditions. Only when the interference is very high (nodes are continuously sending packets), using a block size of 20 bytes results in significantly superior throughput. This suggests that for most error conditions, block size adaptation will not be critical to the performance of Seda.

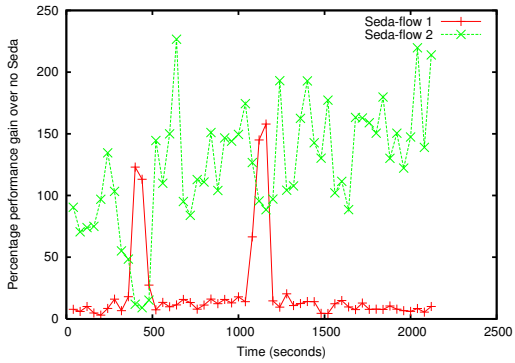

Figure 21. Percentage performance gain vs. Time for two Seda flows

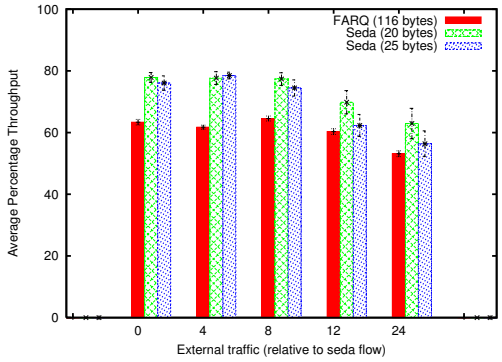

Figure 22. Throughput vs. External traffic with respect to the Seda flow

External load (× Seda traffic)	% performance gain Seda-25	% performance gain Seda-20
0	20.02	22.9
4	27.11	25.78
8	15.35	19.83
12	3.32	15.78
24	6.07	18.42

Table 5. Percentage performance gain in throughput achieved using Seda, compared to the baseline throughput of FARQ, for different external interference loads

For the above experiment, Table 6 shows that Seda with block sizes of 20 and 25 bytes reduces the percentage retransmission traffic by more than 50% compared to FARQ, for different interfering loads.

To study the performance of Seda under increasing system load, we conducted an experiment by varying the rate of transmitting data. We measured the throughput achieved by Seda and FARQ at the rates of 2, 2.5, 3.3, and 10 packets per second, and plot this in Figure 23. We observe that the throughput decreases as the system load is increased, but the decrease is graceful, showing that Seda performs well under higher system loads. A similar observation is seen in the case of the FARQ scheme too.

Table 7 shows the performance gain in throughput achieved using Seda, compared to the baseline throughput

External load (× Seda traffic)	% reduction in retx. traffic Seda-25	% reduction in retx. traffic Seda-20
0	53.84	65.43
4	78.8	57.97
8	47.46	64.83

Table 6. Percentage reduction in retransmission traffic for Seda-20 and Seda-25 over FARQ, for different external interference loads

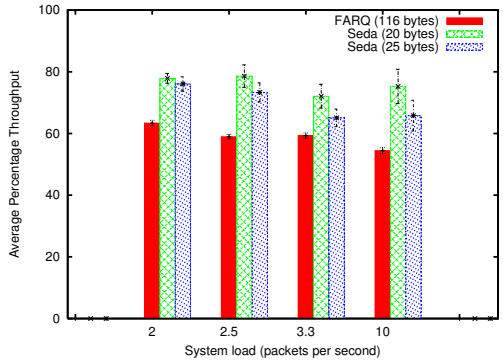

Figure 23. Throughput vs. System load

of FARQ, for different number of packets transmitted every second obtained from Figure 23. We notice that the performance gain in throughput is consistently greater than 20%.

For different source transmission rates, Table 8 compares the average time taken by Seda and FARQ to receive a packet correctly and forward it to the application layer. The time taken to receive a packet completely on an average, is termed as the reception rate. Over a duration of 30 minutes, we measured the number of packets that were received correctly for both Seda and FARQ. Dividing the total number of packets received correctly by the total duration of 30 minutes, we obtain the delay to receive a packet on an average. We observe that the reception rate of Seda is consistently better than that of FARQ. This also suggests that the buffer requirements of Seda will be lower than that of the FARQ scheme.

7 Related Work

Alternate to Seda's ARQ-based error recovery, forward error correction (FEC) is often applied for the same purpose, especially in synchronous wireless networks where large frame delay jitter due to retransmissions cannot be tolerated. However, effective FEC relies on the knowledge of the current BER. In asynchronous wireless networks, especially those defined in unlicensed frequency bands, the channel quality changes very quickly and unpredictably [10, 2, 13, 6]. Thus it is very difficult to gauge the FEC encoding redundancy to match the current channel error rate. Moreover, our measurements (see Section 4.1) in MicaZ motes and those [13] in 802.11 networks show that the bit errors are highly clustered and bursty. Complex interleaving over large block of data must be applied in conjecture with FEC, adding to the computation overhead against low-end low-power wireless transceivers. Also, note that performing FEC at the block

System load (pkts per s)	% performance gain - Seda-25	% performance gain - Seda-20
2	20.03	22.9
2.5	24.4	32.08
3.3	9.68	21.53
10	20.87	38.25

Table 7. Percentage performance gain in throughput achieved using Seda, compared to the baseline throughput of FARQ, for different system loads

Tx rate (ms per pkt)	Seda-25 rx rate (ms per pkt)	FARQ rx rate (ms per pkt)
100	122.6	158.4
300	371.84	435.98
400	439.9	585.05
500	530.22	680.27

Table 8. Comparison of reception rates for Seda and FARQ for different source transmission rates

level will have the same defects as FEC at the frame level.

Recent work [13, 6] has successfully applied packet combining [16, 3] and hybrid ARQ [12, 5] in asynchronous wireless networks. However, SPaC [6] still operates at the frame level, retransmitting encoded versions of a corrupted frame. MRD [13] exploits multiple erroneous versions of the same packets received at multiple spatially redundant 802.11 access points. Seda is similar to MRD in the sense that the receiver treats the received frame as a sequence of blocks. However, no fine-grained error detection codes are available in MRD frames. MRD has to exhaust all block combinations in a number exponential to the number of blocks, in its search for the one that is error-free. More importantly, MRD retransmits the entire frame when the combination fails, therefore shares the same weakness as those frame-level ARQ mechanisms applied in 802.11 [9], TinyOS [8], and SPaC [6]. Both SPaC and MRD explore cooperative communication. This is similar to Seda cooperative streaming and we do not claim novelty in this regard.

Seda borrows the well-known concept of reliable streaming from Internet transport design, TCP in specific. The goal of streaming in Seda, however, is completely different from that in TCP. Seda streaming decouples framing from retransmissions to resolve the conflict between effective error recovery and high channel utilization in asynchronous wireless networks, while TCP streaming enables extremely flexible framing to fit into the a variety of maximum transmission units of different underlying link layers. With Seda streaming, the wireless frame can be made sufficiently long, subject to the available buffer space at the transceiver firmware and the delay constraints of the applications.

8 Conclusion

The fundamental issue that limits the achievable throughput in a sensor network with low-end, low-power wireless transceivers stems from the fact that datalink layer framing cannot achieve both effective error recovery and high channel utilization. This paper describes our design and imple-

mentation of Seda, a reliable datalink layer that decouples framing from error recovery. Seda introduces a new block abstraction, with block size chosen for optimal error recovery. Meanwhile, Seda relaxes the bound on the maximum frame size and makes the frame error rate irrelevant to framing. As a result, the frame can be sufficiently large to ensure high utilization of the wireless channel. Our evaluations through intensive simulation and real experiments have confirmed the benefits of Seda. Seda improves the throughput by at least 10% and up to 25% when the channel quality is low. Further, Seda reduces the percentage retransmission traffic by more than 50% compared to a frame-based retransmission scheme. The design of Seda also exposes that future sensor motes should be equipped with radios with more packet buffer space to achieve optimal utilization of the channel capacity. We are currently extending the basic cooperative Seda streaming with power control, which potentially leads to further capacity conservation for high-performance wireless communications in wireless sensor networks. We are also studying the impact of Seda on the well-known tradeoff between throughput and latency in wireless networks.

Acknowledgments

This work was supported in part by NSF grants CNS-0626825, CNS-0615318, CNS-0509233, and CCF-0208769. Any opinions and findings expressed in this paper are those of the authors and do not necessarily reflect those of the funding agencies.

9 References

[1] CC2420 Data Sheet. `http://www.chipcon.com/`.

[2] D. Aguayo, J. Bicket, S. Biswas, G. Judd, and R. Morris. Link-level measurements from an 802.11b mesh network. In *Proceedings of ACM SIGCOMM*, 2004.

[3] S. Chakraborty, E. Yli-Juuti, and M. Linaharja. An ARQ scheme with packet combining. *IEEE Communications Letters*, 2(7):200–202, July 1998.

[4] Crossbow Technologies. http://www.xbow.com/.

[5] A. Daraiseh and W. Baum. Methods for packet combining in harq systems over bursty channels. *ACM Mobile Networks and Applications*, 2:213–224, October 1997.

[6] H. Dubois-Ferrière, D. Estrin, and M. Vetterli. Packet combining in sensor networks. In *Proc. of ACM SenSys*, 2005.

[7] J.-P. Ebert and A. Willig. A gilber-elliot bit error model and the efficient use in packet level simulation. Technical Report TKN-99-002, Technische Universität, Berlin, March 1999.

[8] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister. System architecture directions for networked sensors. In *Proc. of ASPLOS*, pages 93–104, November 2000.

[9] IEEE Computer Society. Wireless LAN medium access control (MAC) and physical layer (PHY) specifications. IEEE standard 802.11, 1999.

[10] A. P. Jardosh, K. N. Ramachandran, K. C. Almeroth, and E. M. Belding-Royer. Understanding link-layer behavior in highly congested ieee 802.11b wireless networks. In *Proc. of SIGCOMM Workshop on Experimental on Experimental Approaches to Wireless Network Design and Analysis (E-WIND)*, 2005.

[11] P. Levis, N. Lee, M. Welsh, and D. Culler. Tossim: Accurate and scalable simulation of entire tinyos applications. In *SenSys '03: Proc. of the 1st International Conference on Embedded Networked Sensor Systems*, pages 126–137, 2003.

[12] S. Lin, D. Costello, and M. Miller. Automatic-repeat-request error-control schemes. *IEEE Communications Magazine*, 22(12):5–17, December 1984.

[13] A. K. Miu, H. Balakrishnan, and C. E. Koksal. Improving loss resilience with multi-radio diversity in wireless networks. In *Proc. of ACM MobiCom*, pages 16–30, Cologne, Germany, September 2005.

[14] J. Polastre, J. Hui, P. Levis, J. Zhao, D. Culler, S. Shenker, and I. Stoica. A unifying link abstraction for wireless sensor networks. In *SenSys '05: Proc. of the 3rd international conference on Embedded networked sensor systems*, pages 76–89, 2005.

[15] I. Rhee, A. Warrier, M. Aia, and J. Min. Z-mac: a hybrid mac for wireless sensor networks. In *SenSys '05: Proc. of the 3rd international conference on Embedded networked sensor systems*, pages 90–101, 2005.

[16] P. Sindhu. Retransmission error control with memory. *IEEE Transactions on Communications*, 25:473–479, May 1977.

[17] T. van Dam and K. Langendoen. An adaptive energy-efficient mac protocol for wireless sensor networks. In *Proc. of the 1st International Conference on Embedded Networked Sensor Systems, Sensys '03*, pages 171–180, November 2003.

[18] A. Woo, T. Tony, and D. Culler. Taming the underlying challenges of reliable multi-hop routing in sensor networks. In *Proc. of ACM SenSys*, pages 14–27, Los Angeles, CA, November 2003.

[19] W. Ye, J. Heidemann, and D. Estrin. An energy-efficient mac protocol for wireless sensor networks. In *Proc. of IEEE Infocom*, pages 1567–1576, June 2002.