Hindawi Publishing Corporation
Journal of Computer Systems, Networks, and Communications
Volume 2009, Article ID 409853, 14 pages
doi:10.1155/2009/409853

Research Article

On Reliable Transmission of Data over Simple Wireless Channels

Pawel Gburzynski,[1] Bozena Kaminska,[2] and Ashikur Rahman[2]

[1] *Department of Computing Science, University of Alberta, Edmonton, AB, Canada T6G 2E8*
[2] *School of Engineering Science, Simon Fraser University, Burnaby, BC, Canada V5A 1S6*

Correspondence should be addressed to Pawel Gburzynski, eurasip.pawelg@sfm.cs.ualberta.ca

Received 1 August 2008; Accepted 4 January 2009

Recommended by Y.-M. Huang

Standard protocols for reliable data transmission over unreliable channels are based on various Automatic Repeat reQuest (ARQ) schemes, whereby the sending node receives feedback from the receiver and retransmits the missing data. We discuss this issue in the context of one-way data transmission over simple wireless channels characteristic of many sensing and monitoring applications. Using a specific project as an example, we demonstrate how the constraints of a low-cost embedded wireless system get in the way of a workable solution precluding the use of popular schemes based on windows and periodic acknowledgments. We also propose an efficient solution to the problem and demonstrate its advantage over the traditional protocols.

1. Introduction

Many wireless sensing devices can comfortably operate one way, that is, sending their samples at some intervals with no feedback from the receiving end, assuming that occasional losses are acceptable. There exist, however, a few sensing applications where all samples are considered important. They involve systems where the samples represent a certain process to be analyzed at the receiver, and the fidelity of that analysis is critical. Many areas of medical monitoring, especially those dealing with tracing and diagnosing heart activity, fall into this category. Even though one may argue that occasional gaps in the sampled data can be filled by interpolation or other "guessing," the community of health care professionals is not receptive to such arguments. On top of the understandable obsession about the utmost quality of data constituting the basis for life-saving diagnosis, medical procedures are prone to (often uninformed) public criticism and litigation. To prevent it, the vocabulary of terms characterizing the precision of medical diagnostic procedures must exclude phrases like "almost all data" and "approximate records."

From the engineering point of view, one would like to build a practical device with the minimum cost, where by "practical," we understand one that works and fulfills the expectations of its users. Even if those expectations are high, an overdesigned device is bound to cost more than one that meets those expectations with the minimum expense of resources, be it memory, CPU power, or RF bandwidth. Notably, the amount of RF bandwidth needed by the device affects more than just the monetary cost of the project. The RF spectrum is bound to be more and more polluted everywhere, and especially so in places like health care facilities, where the multitude of wireless sensors will have to compete for bandwidth to deliver all the samples to their collection devices. So, we cannot just say that money is no object for the kind of reliability requirements inherent in medical applications, and for example, nonchalantly overdesign the device for bandwidth. Our goal should be rather to find the right set of algorithms and protocols to accomplish the reliability objectives with the minimum bandwidth possible. In addition to reducing the raw cost of the device, this approach will also result in an "environmentally friendly" design, and if only to our own immediate advantage, will allow us to deploy more devices within a given perimeter.

The problem addressed in this paper arose during the design of a wireless sensing device for heart monitoring based on ballistocardiography (BCG [1, 2]). The primary function of the device, dubbed HDL in the sequel (for Heart Data

Logger), was to collect data samples from a set of sensors, store them temporarily in local flash memory, and transmit them reliably (in near real time) to a workstation (which we will call the CPP) over a wireless channel. (For collection and processing point.) Cost considerations combined with restrictions regarding the RF bandwidth, as well as demands for long sustained battery operation, led us to base the wireless link on a cheap low-power RF device driven by a microcontroller.

The most challenging element of the design was the protocol for transmitting the sampled data to the CPP. Even ignoring the losses, the amount of bandwidth needed to transmit the samples in real time approached the physical capability of the RF module. A by-the-book implementation of a two-way window-based ARQ scheme with periodic (sparse) acknowledgments and retransmissions [3] brought the system down to its knees. Regardless of how selective the acknowledgments were, the very fact that the transmitter had to expect feedback and make room for it within the stream of outgoing data rendered its operation extremely inefficient. Consequently, we have devised and studied alternatives to those obvious and popular schemes and arrived at a solution meeting our objectives.

Besides addressing a specific problem, our paper demonstrates that the realm of small embedded wireless systems brings about a bag of idiosyncratic constraints which often force us to look for solutions off the beaten path. Unfortunately, the high-level approach to transport and application-layer protocols, pervading most academic research, tends to ignore the kind of mundane low-level implementation constraints that proved decisive in our study. To a large extent, this is yet another fallout from protocol layering, which has met with consistent criticism in the world of wireless communication [4–9]. One has to resort to cases of product engineering to show those issues in the proper light of their highly practical relevance.

2. The System

2.1. General Outline. Ballistocardiography [1, 2] is a method of collecting and interpreting data about heart action by measuring the acceleration of body surrounding the heart area. The acceleration is detected and measured by sensors attached to the body and transformed into digital data samples, which are subsequently analyzed and visualized by DSP software. In our design, the accelerometers are connected (by wires) to the HDL device, which is responsible for the analog-to-digital conversion, intermediate storage of the data, and its transmission to the CPP for analysis and visualization.

The device is equipped with a small amount of flash memory which plays a dual role. From the viewpoint of transmitting BCG samples to the CPP, it acts as a buffer compensating for the transmitter's jitter and allowing the node to retransmit missed samples. It also functions as a simple database storing the last few sample streams taken from the subject, which can be transmitted retroactively upon request from the CPP.

2.2. Hardware and Software. The essential hardware components of the HDL are the MSP430F1611 microcontroller [10] and the CC1100 RF module [11] (both by Texas Instruments). RF band restrictions prevented us from using Bluetooth for the radio link (the 916 MHz band was practically the only option), although a variant of the HDL utilizing a Bluetooth module was built and tested. Another reason for rejecting Bluetooth was the considerably larger hardware cost as well as significantly increased power requirements. Moreover, the arcane rules for device pairing and the consequent long and nondeterministic delays in binding the HDL to its CPP (especially with other Bluetooth devices present in the neighborhood), turned out to be prohibitively troublesome.

The microcontroller was programmed under PicOS [8, 12–14], which is a convenient and highly efficient operating system for small-footprint devices, capable of structural multithreading within the confines of the tiny RAM available on low-end microcontrollers. In fact, MSP430F1611 is the largest representative of the MSP430 family, with 10 KB of RAM, which turned out to be more than needed. Our primary concern was to provide an RAM buffer for accommodating the samples before storing them in flash, to account for the occasional hiccups caused by the various special conditions, for example, the need to erase before write on the boundary of a nonempty block. To maximize the life of the flash memory, we avoid unnecessary erase operations and balance the usage of all its segments. Consequently, the procedures (system calls) writing data to flash may block awaiting the moment when the segment to be written gets into the proper state.

The input to the HDL consists of eight analog signals arriving from the accelerometers. Those signals are fed to the eight ADC ports of MSP430F1611 and converted into 8 12-bit values at the rate of $f_s = 500$ conversions per second, yielding $8 \times 12 \times 500 = 48000$ bits per second of incoming data. That rate is a required standard for a diagnostic-grade digital representation of the BCG signal. To avoid confusion, from now on, by a *sample* we will mean a single set of 8 values collected every 1/500 of a second, while the complete series of samples sent for processing to the CPP will be called a *take*. In other words, a take represents a complete measurement, which is processed and visualized at the CPP for the purpose of assessment or diagnosis.

The ADC converter as well as the sample collection process are turned on upon a request from the CPP. Such a request specifies the duration of the take in seconds, which is transformed by the HDL into the corresponding number of samples. Those samples are then collected and stored in flash memory. They can be transmitted in parallel with their collection, or merely collected and stored locally to be transmitted later. The device employs a simple differential compression scheme applied at the stage when the 12-bit ADC samples are repackaged into 48-byte blocks, which are the storage/transmission units. Owing to the nature of the BCG data, the compression scheme brings about highly consistent 45% average savings, which practically never get below 42%. This means that a typical 48-byte block accommodates over 7 samples. As with all loss-less

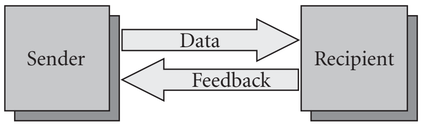

Figure 1: The communication model.

compression techniques, it is theoretically possible (although unimaginable in practice) that the scheme will spontaneously inflate the data size by up to 29%.

The range of data transmission rates available to CC1100 is adjustable within a certain interval, with the raw limit around 200 kbps, which, considering the coding (Manchester), framing, and interpacket spacing reduces to about 50 kbps of effective rate (assuming one-way back-to-back packets).

3. Data Communication

3.1. The Problem and Its Classical Solutions. The amount of bandwidth needed to maintain a connection between an HDL and its CPP is trivial, except for the transmission of takes from the HDL to the CPP. Thus, we will focus on this highly asymmetric communication scenario, whereby a significant amount of data is transferred essentially in one direction. The classical problem of reliable data transmission over an unreliable link is formulated in the context of two parties, one of them being the sender (S) and the other the recipient (R), as shown in Figure 1. The setup assumes two separate (possibly logical) channels. With the reverse channel, the recipient is able to convey feedback to the sender, for example, to request retransmission of the missing (damaged) packets.

The simplest solution to the problem has been known as the alternating bit protocol (ABP) [15, 16], and consists in acknowledging every single received packet by the recipient. Generalizations and improvements upon this simple scheme are known under the generic name of ARQ protocols [3]. Their objective is to reduce the amount of traffic in the recipient-to-sender direction, and provide for smooth operation in the face of nontrivial propagation delays between the two parties [17]. They typically involve a *window*, representing the limit on the number of outstanding (i.e., sent but unacknowledged) packets or bytes that the sender is allowed to send ahead. In the simplest case, a positive acknowledgment referring to a specific packet indicates that all packets up to and including the acknowledged one have been received [18]. Some schemes employ negative acknowledgments [19, 20] to explicitly describe what has been missing, some others rely solely on timeouts. With the latter, having received no acknowledgment for an excessive amount of time, the sender will begin retransmitting packets from the first unacknowledged one [19].

3.2. System Conditioning. The most difficult problem facing an ARQ scheme in our system is the utmost simplicity of the radio link and the lack of a reasonable reservation mechanism that would allow us to implement reliable and deterministic separation of the two logical channels shown in Figure 1. Based on our estimates in Section 2.2, the available bandwidth is already close to the minimum required to sustain the data transfer alone. Consequently, any attempts to impose extra structure on that bandwidth (e.g., acknowledgment slots akin to 802.11 [21–24]), while possibly facilitating orderly delivery of data packets, would significantly reduce the amount of bandwidth available to those packets.

CC1100 comes equipped with rudimentary tools for collision avoidance [11]. The module is able to recognize radio activity in the neighborhood, based on a definable threshold, and automatically hold its own transmission until the activity is gone. The system can take advantage of this function to implement a medium access control scheme facilitating coexistence of multiple nodes within their mutual range. Owing to the fact that our design admits such a situation, we would like, as much as possible, to provide for a social behavior of the multiple HDLs present in the same area. Realistically, we cannot hope to achieve more than one take transfer at a time. However, we should be able to have multiple HDLs reporting their status to the CPP and responding to simple requests effectively in parallel.

To this end, our driver of the RF module implements a simple listen before transmit (LBT) scheme, whereby a node perceiving a radio activity before transmission backs off to avoid a collision. Such a scheme greatly facilitates low-bandwidth communication among multiple nodes, but it does waste bandwidth. Note that all request and status messages are very short. Thus, complex handshakes of the RTS-CTS-DATA-ACK variety are completely useless (and would be harmful [25]) under the circumstances.

Under ideal conditions, that is, no interference from other nodes, the HDL transmitter is able to send packets back-to-back reasonably fast, with minimal interpacket spacing. However, as soon as the collision-avoidance mechanism kicks in (i.e., foreign activity is sensed and backoff is employed), it “loses its step” and may remain blocked for a relatively long time. This is because the time intervals of the collision-avoidance scheme are measured in tens of milliseconds; the RF module is slow to respond to the changes in its status and exhibits a considerable inertia in its built-in LBT mechanism. Besides, the smallest sensible backoff window is about 20 milliseconds.

3.3. Data Formats. Data exchanged between the HDL and its CPP consist of packets framed as shown in Figure 2. This layout, including the maximum payload (data) length, is enforced by the physical characteristics of CC1100. While theoretically, by resorting to some tricks, it would be possible to have payloads of arbitrary length, the 55-byte maximum for the *data* component is already on the large side, considering the reliability of single-packet reception. (CC1100 uses a 64-byte internal FIFO for storing the outgoing/incoming frame. When the total length of the packet exceeds the FIFO size, the packet must be sent and received “in pieces,” which is not very practical.)

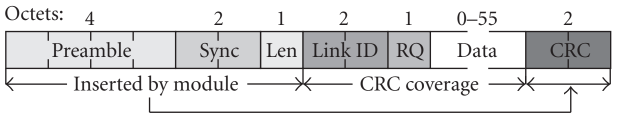

Figure 2: The frame format.

The first logical component of a packet is the link ID, that is, the temporarily unique session identifier assigned by the CPP to one particular HDL. This field is used to tell apart multiple HDLs supervised by the same CPP. The single-byte RQ field identifies the packet type, that is, the kind of request or response carried in "data."

A packet representing a take fragment always carries a block of 48 bytes encoding a compressed portion of successive samples (see Section 2.2). It also includes the block number, such that its position within the take can be determined independently of other blocks. The data portion of such a packet consists of 52 bytes partitioned between the block number and the 48-byte chunk of compressed data.

3.4. Traditional ARQ Schemes. The first transmission protocol that we tried in our design operated as follows. When the CPP wants to retrieve a take from an HDL, it sends to the HDL a short SEND request that, in addition to the take identifier, specifies two parameters: F—the starting block number, and N—the number of blocks to be retrieved. The same request format is also used as a positive or negative acknowledgment. Initially, to start the retrieval, the CPP sends a request with $F = 0$ and N equal to the total number of blocks in the take.

Having received the initial SEND request, the HDL starts to transmit the requested blocks consecutively. As the block number is included in every packet, the receiving node can tell which blocks have made it and which have been lost.

Conceptually, when the CPP receives a new block of data that adds to the continuous sequence of already received blocks belonging to the requested take, it should *acknowledge* the reception with a new SEND request specifying the next expected block number. A block number behind the last block of the sample means that the entire take has been received. This is how the simplest window-less version of the scheme could work.

To avoid too many acknowledgments, the HDL is allowed to use a window, that is, having sent the first block, it is permitted to send a few blocks ahead without waiting for separate requests for them. Specifically, the HDL maintains two counters: *NextToGo* and *NextToAck*. The first counter tells the number of the next block to be transmitted, while the second one points to the first block for which an acknowledgment (meaning an SEND request) has not been received yet. The device is allowed to keep sending blocks for as long as $NextToGo - NextToAck < W$, where W is the assumed window size. To facilitate this operation, the CPP will refrain from acknowledging every single block. Ideally, it would like to get away with a single acknowledgment per the entire window.

Having reached the end of window, for as long as *NextToAck* is not advanced, the HDL keeps retransmitting the last block (at some reasonably short intervals). If the CPP has missed some packets from the window, it should repeat the last SEND message until the missing fragment arrives. A duplicate SEND request for the *NextToAck* block is viewed by the HDL as a request to retransmit all packets starting from *NextToAck*. Note that as soon as the hole (or holes) in the received data have been plugged by the CPP, it can issue a SEND message whose offset will jump to the end of the received continuous set. This will tell the HDL to abandon the retransmission and move ahead.

In general terms, the described scheme falls into the standard family of Go-Back-N ARQ protocols and, in particular, lies at the foundation of TCP [26]. The key to its effectiveness in the application at hand is in the right selection of the window size W, the intervals between SEND messages, and the timeouts. Notably, the HDL has to make sure that the acknowledgments can be received at all; thus, it cannot be overly aggressive with transmissions.

Note that the problem is quite idiosyncratic of our RF framework. In the classical analysis of the ARQ schemes, it is commonly assumed that the cost of receiving an acknowledgment has nothing to do with the cost of sending a data packet. The primary role of the window in such a system is to compensate for the end-to-end propagation delay by turning the channel into a pipe [27]. In our case, the propagation delay is insignificant, and the window is used as the grain of acknowledgment—to avoid too frequent "channel reversals" and interruptions in the "proper" stream of data arriving from the HDL.

To the best of our knowledge, the problem of implementing an ARQ scheme over a wireless channel has never before been looked at from this particular angle. Most of the previous efforts in this area have focused on two issues: (1) adapting TCP for wireless connections [28–30], and (2) lowering the effective packet error rate in cellular channels [9, 31, 32]. In the second case, the role of ARQ is not to absolutely guarantee the delivery of all data, but to reduce the number of losses, possibly in combination with various forward error correction (FEC) techniques naturally employed in cellular systems [9]. All those variants of ARQ assume that the requisite acknowledgments are delivered over a separate channel whose interference with the primary (data) channel is either completely immaterial, or its nature is much less destructive than with our "channel reversal."

The amount of time needed to expedite a single data packet containing a block of the requested take is $t_p \approx 5$ milliseconds. Adding to this the various unavoidable delays in the OS and in the driver, we conclude that with back-to-back transmissions we are able to send about $f_p \approx 150$ blocks per second. Considering that the frequency of sampling f_c is 500 samples per seconds, which translates into about 70 blocks on the average (see Section 2.2), we find ourselves comfortably within the realm of feasibility. One should realize, however, that this optimistic conclusion only holds under the assumption that the stream of transmitted data is not disrupted for excessively long intervals. As it turns out, any attempts to provide reception opportunities (LBT) between a pair of sent data packets incur a time overhead t_a of order 15–30 milliseconds, which is 3–6 times more than the transmission time of one block.

Consequently, the performance of the traditional ARQ scheme, in the version described above, was pathetic regardless of the setting of its parameters. With the LBT delays and the extra losses incurred by the interference from acknowledgments (backoffs), the effective rate went down to about 15–20 blocks per second. This way, the amount of time needed to transmit a 20-second take was close to two minutes.

Most of the relevant insight into the problem can be obtained from a very simple model based on the following assumptions:

(1) the cost (time) of a packet (block) transmission within a window is fixed and equal t_p;

(2) the time penalty of receiving a feedback from the recipient is also fixed and equal to t_a;

(3) the probability of a packet error, expressed as P_e, is fixed and independent.

Let W denote the window size. With sparse acknowledgments separated as widely as the window size, the approximate amount of time needed to transmit a take of N blocks can be expressed as

$$T(N) = W_c \times t_p + t_a + \sum_{i=0}^{W_c} P(i, W_c) \times T(N-i), \tag{1}$$

where $W_c = \min(W, N)$ and

$$P(n,m) = \begin{cases} P_e \times (1-P_e)^n & \text{if } n < m, \\ (1-P_e)^m & \text{otherwise} \end{cases} \tag{2}$$

is the probability that exactly n initial blocks of m total have been transmitted successfully. This yields the following recursive formula:

$$T(N) = \frac{W_c \times t_p + t_a + \sum_{i=1}^{W_c} P(i, W_c) \times T(N-i)}{1 - P(0, W_c)}, \tag{3}$$

with $T(0) = 0$.

If the acknowledgments are in fact sent independently (based on loose timeouts at the recipient), t_p has to be large enough to provide a reception opportunity after every packet. On the other hand, one can try to make the windows explicit and formally request that the feedback be only sent at the window boundary. This may require special signals (short packets) at the end of a window—to notify the recipient that the feedback is expected and should be provided. While those "signals" may take more time than a simple reception opportunity for an acknowledgment, the packets sent within a window can be spaced tightly (no LBT) avoiding much of the overhead. The advantage is illustrated in Figure 3, which compares the normalized time of transmitting a long series of blocks under different scenarios, with the two variations discussed above represented by curves I. and II. The normalized time is expressed as the ratio of the

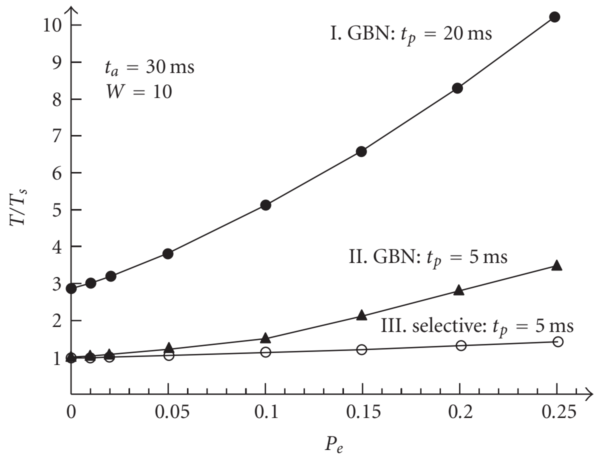

Figure 3: Normalized time of sending a series of blocks under Go-Back-N and selective schemes.

actual transmission time T to the time T_s required to collect N samples, where N is the number of transmitted packets (thus, any value above 1 should be viewed as a failure to keep pace with the sample collection process). The window size was 10 blocks. In the first (spontaneous) case, t_p is set to 20 milliseconds, which roughly corresponds to the minimum reasonable packet separation interval that would provide any reception opportunities at all. The cost of receiving an acknowledgment was set to 30 milliseconds in both cases, which was rather optimistic. No detailed experimental tests were carried out for these schemes, as they were immediately seen to be practically useless.

Indeed, once we conclude that acknowledgments should only be sent at explicit window boundaries, it makes little sense to follow the Go-Back-N approach. Instead, it would be much wiser to selectively indicate in the acknowledgment the exact blocks that were missing in the window. Then, the next window would start with the missing blocks from the previous one. The performance of this new scheme is captured by (1) substituting

$$P(n,m) = \binom{m}{n} \times (1-P_e)^n \times P_e^{(m-n)}, \tag{4}$$

which represents the probability that exactly n of the m blocks have been received correctly.

The solution for $W_c = 10$ is depicted by curve III. in Figure 3. Its superiority over the Go-Back-N approach is clear. One of its desirable features is the strict monotonicity with respect to the window size, which is not the case with the Go-Back-N scheme. This is because, with Go-Back-N, an error occurring within a large window will trigger the retransmission of the entire tail. While a jump ahead can be forced by the CPP when it detects that the hole has been plugged (this possibility is not captured by our simple model), large windows will tend to contain multiple erroneous packets, which will nullify the impact of this feature.

The difference in character between a Go-Back-N protocol and a selective scheme applied to our system is shown

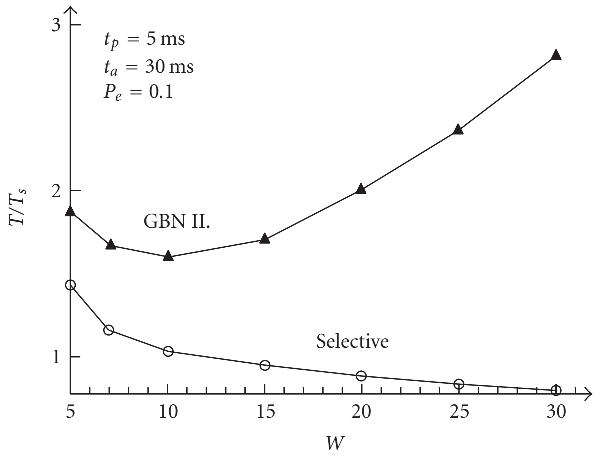

Figure 4: Behavior of Go-Back-N and selective schemes for different window size.

in Figure 4. Parameters t_p and t_a correspond to the values characteristic of our platform, and the probability of error $P_e = 0.1$ is relatively large, but not uncommon in our application.

3.5. The Workable Scheme. The monotonicity of the simple selective scheme with respect to the window size W (Figure 4) suggests that the window size should be as large as possible. Note, however, that one factor tacitly ignored in the simplified performance model is the size of the feedback (acknowledgment) packet, which depends on the number of missing blocks. One would like to avoid a situation when the feedback message itself consists of multiple packets because then we would have to cope with two more issues.

(1) Multiple channel reversals, which, as we explained in Section 3.2 tend to waste a disproportionate amount of bandwidth.

(2) Reliable reception of the feedback. In a situation when the feedback message consists of multiple packets, simple persistent and idempotent schemes will not work, which will bring about more bandwidth wastage.

As a side note, let us mention here a third issue, usually ignored in protocol design studies, namely, the complexity of the implementation expressed in the mundane terms of code length. This parameter is not irrelevant in the realm of microcontrollers; even if the computational complexity of a program is acceptable, the program should not be too big, as it may not fit into the limited ROM.

As Figure 4 shows diminishing returns for pushing the window size beyond a "reasonably" large value, we propose to make the window size variable and adopt the following set of rules:

(1) the window size is determined by the (maximum) number of requested blocks whose description can fit into a single request/feedback packet;

(2) the end of a window is explicitly indicated by the sender (the HDL) in a persistent manner until noticed by the recipient (the CPP);

(3) Request packets are idempotent and they are sent persistently by the CPP, until it notices the arrival of a new window.

This approach minimizes the number of channel reversals and results in a scheme whereby the CPP repeatedly polls the HDL for the missing blocks and then expects to receive them in the next window of packets, doing so until all the blocks have made it. At every step, the CPP asks for the maximum number of blocks that can be described in a single request packet. The generic algorithm for take extraction executed by the HDL can be described as follows.

(1) *Wait for a request.* This is the main loop executed by the HDL while being idle. *Having received a request, proceed at 2.*

(2) *Extract from the request the list of blocks to be sent and send them blindly back-to-back (no LBT) until the last requested block has been expedited. Then proceed at 3.*

(3) *Send periodically, at reasonably sparse intervals and with LBT on (as to enable reception opportunities), a short packet indicating the end of window. A natural choice for such a packet is an empty block. Keep doing so until a new request arrives from the CPP. Then proceed at 4.*

(4) *If the request is NULL, that is, no more blocks are needed, terminate the operation and proceed at 1. Otherwise, proceed at 2.*

A flowchart view of the above algorithm is shown in Figure 5, with the ovals representing waiting states, and the triangle (labeled Rq) indicating the event consisting in the reception of a request packet. For transmission while sampling, the algorithm starts with one implicit round whereby the blocks are sent back-to-back while being collected from the ADC. This constitutes the first window consisting of all blocks contributing to the take. Following that round, the algorithm continues at step 3.

The CPP's end of the protocol looks like the following.

(1) *Initialize by marking all blocks to be received as absent. Then continue at 2.*

(2) *Find out which blocks are still absent. If none, send a NULL request and enter the IDLE state. Otherwise, prepare a single request packet that covers as many of the missing blocks as possible and proceed at 3.*

(3) *Keep sending the request packet at reasonably sparse intervals until a block packet arrives from the HDL. Then proceed at 4.*

(4) *Keep receiving the blocks and storing them until you see the end-of-window packet (an empty block). Then proceed at 2.*

Figure 6 shows the flowchart view of the above algorithm. The Bk triangle represents the reception of a block packet

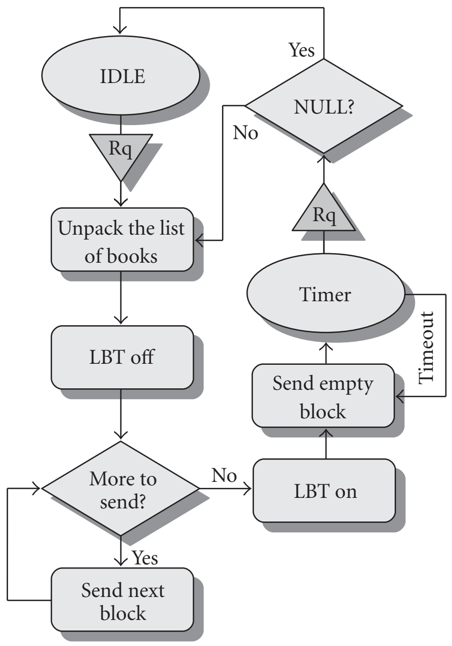

Figure 5: The transmission part of the protocol.

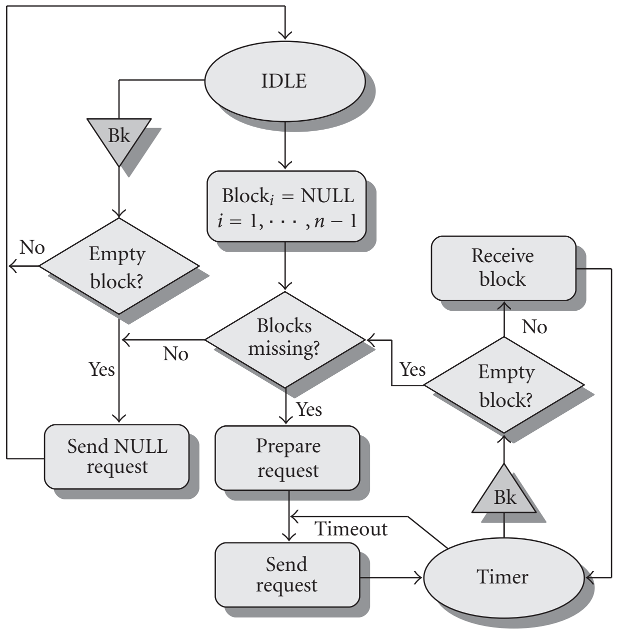

Figure 6: The reception part of the protocol.

from the other party. For an immediate transmission-while-sampling request, the first request issued by the CPP will instruct the HDL to initiate the sampling procedure, and will implicitly trigger the first “total” window consisting of all the blocks contributing to the take.

The simple (idempotent) nature of requests and block transmissions automatically takes care of all possible loss scenarios and races. Note, for example, that all blocks of a given window requested by the CPP can be lost. Thus, when the HDL arrives at step 3 of its algorithm, it will eventually receive again the original request, which persists at the CPP until it sees the first (any) packet of the requested window. Consequently, it will just retransmit the last requested window in its entirety. Similarly, the CPP does not assume that its single NULL request (sent in step 2 to indicate the completion of the transfer) always makes it to the HDL. Rather, having assumed the IDLE state, it will reply with a NULL request to any end-of-round packet received from the HDL, to eventually force it to the IDLE state as well. This is illustrated in Figure 6 with the (spurious) Bk event in the upper left area of the flowchart.

3.6. Request Formats.

3.6. Request Formats. The key to a good performance of the protocol described in the previous section is the efficient description of missing blocks in the request packet. We have implemented two formats of such requests.

With format 1, a request packet is filled with block numbers, each number occupying three consecutive bytes, up to the maximum of 16 values (48 bytes). A simple trick is played to describe individual blocks as well as continuous ranges. Suppose that $c_0, \ldots, c_{n-1}$ is the sequence of block numbers specified in a request packet. These numbers are interpreted by the HDL as follows.

(1) Set $i = 0$.

(2) If $i = n$, done (all requested blocks have been sent). Otherwise, set $c_a = c_i$ and $i = i + 1$.

(3) If $i = n$ or $c_i > c_a$, send block c_a. Otherwise, set $c_b = c_i$ and $i = i+1$. Send the blocks $c_b, \ldots, c_a$. Continue at 2.

Thus, for as long as the block numbers are increasing, they describe individual blocks, while a decreasing number together with its preceding number are taken together as the boundary of a continuous range (chunk) of blocks. Note that the pair $N - 1, 0$ (where N is the total number of blocks in the take) requests the (initial) total window comprising the entire take.

With format 2, it is possible to use bit maps to efficiently describe sizable hollow fragments. The collection of requested blocks is described by a sequence of *elements*, which may identify continuous ranges of blocks, as well as random selections represented by bit maps. Each element starts with a header, which consists of one (leading) byte followed by a three-byte block number, as shown in Figure 7. The most significant bit of the leading byte, labeled T, distinguishes between two element types: *origin* ($T = 0$) and *chunk* ($T = 1$). An origin type element requests explicitly the block number *ORG* to be sent to the CPP, and also sets the current location within the take to the block number *ORG* + 1. If *ms* is nonzero, then *ms* consecutive bytes following *ORG* are interpreted as a bit map requesting selected blocks from the take fragment starting at block number *ORG* + 1. This is the block number corresponding to the first bit in the map.

The second element type (chunk) describes a continuous range of blocks starting at the current position, as determined by the previous sequence of elements within the packet. The three bytes following the header byte encode the number of blocks falling into the chunk. If the chunk element

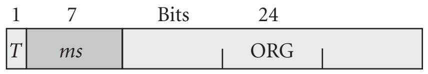

Figure 7: The layout of an element header.

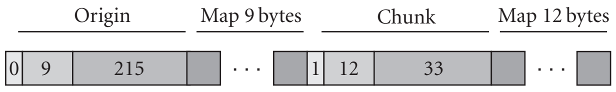

Figure 8: A request fragment.

is the first element of the request, that is, the current position has not been explicitly defined, it is assumed to be zero. Thus, the complete take (all blocks) is described by a single element whose first byte is 0×80 and the following three bytes contain the total number of blocks in the take.

Similar to an origin element, a chunk element may specify a map (its *ms* field may be nonzero). The bits of such a map apply to blocks immediately following the chunk. Figure 8 shows a sample request fragment which calls for block number 215, then uses a bit map to select among blocks 216–287. Note that one byte of the map covers 8 consecutive blocks; thus, the 9-map bytes describe 72 blocks starting at block 216. The subsequent chunk element requests the continuous range of blocks 288–322 (33 blocks total). Finally, the second map applies to blocks 323–418. The total length of the request fragment shown in Figure 8 is 29 bytes.

One can think of several ways to generate requests (in either format). The problem is nontrivial, if we want to do it in an absolutely optimal fashion. For example, it may be OK to request some superfluous (already received) blocks, if the request can be shortened this way. This makes sense when a request that would not fit into a single packet can be thus made to fit. Note that the cost of including a superfluous block in the window (t_p) is relatively low compared to the cost of handling an extra round.

The simple heuristics for format 2 used in our implementation of the protocol in the CPP work this way.

(i) The first missing block is used as the *ORG* of the new request packet.

(ii) Starting from *ORG* + 1, consecutively numbered blocks are examined in bunches of 8 (corresponding to the bytes of the bit map). If at most one block per 8 is present (the map byte contains at most one zero), the bunch becomes a candidate for a chunk. If five or more consecutive bunches (at least 40 blocks) are collected this way, a chunk element is generated and it covers all the subsequent blocks whose 8 bunches contain no more than one superfluous block. Note that such an element takes 4 bytes, that is, less than a map covering five or more bunches.

(iii) If a bunch is all zeros (all the blocks from the bunch are already present), it becomes a candidate for a *skip*, that is, advancement to a new *ORG*. A new *ORG* element is generated whenever we hit five or more consecutive bunches with this property.

(iv) Otherwise, a map is built until either a chunk or a skip is encountered (according to the above criteria), or the request packet is completely filled up. Note that such a map can follow a chunk.

(v) This procedure continues until the request packet is filled completely or there are no more blocks to request. When the round is over, following the reception of the requested window, a new request packet is generated according to the same set of rules.

3.7. Analytical Assessment. The scheme discussed in the last two sections can be viewed as a modification of the selective retransmission protocol described at the end of Section 3.4. The modification consists in making the window size variable. Except for the first round, in which the window covers the entire set of blocks, the size of every subsequent window is determined by the *capacity* of the request packet sent by the CPP, understood as the conveyed number of missing blocks. To estimate that number, suppose, as we did in Section 3.4, that packet errors are independent events occurring with probability P_e. Let us begin with format 1. Assume that we are at the end of the first round and our objective is to fill a request packet of size u slots, where one slot accommodates one block number. The question we ask is as follows : what is the expected number of missing blocks that will be described by such a packet?

In an asymptotically interesting case, we are looking at a large (infinite) number of blocks to transmit, and there are always sufficiently many missing blocks to fill an entire request packet. Moreover, their configuration is nontrivial, that is, not all of them are missing (in which case just two slots would do). Whenever we hit a missing block, there are two possibilities.

(i) The block is a “singleton,” that is, the next block is not missing. This event will occur with probability $1 - P_e$. In such a case, we will use exactly one slot of the packet to describe exactly one block.

(ii) With the remaining probability of P_e, the next block is missing as well, and we have a continuous range of two or more missing blocks. In that case, we will use two slots and the expected number of blocks covered by them is equal

$$\sum_{k=2}^{\infty} P_{k-2} \times k, \tag{5}$$

where P_{k-2} is the probability of exactly $k - 2$ consecutive errors (note that we already know that two consecutive blocks are missing).

Thus, the expected capacity of a format 1 request with u slots is given by the following recurrence relation:

$$C_u = (1 - P_e)(C_{u-1} + 1) + P_e \left(C_{u-2} + (1 - P_e) \sum_{i=2}^{\infty} P_e^{i-2} \times i \right), \tag{6}$$

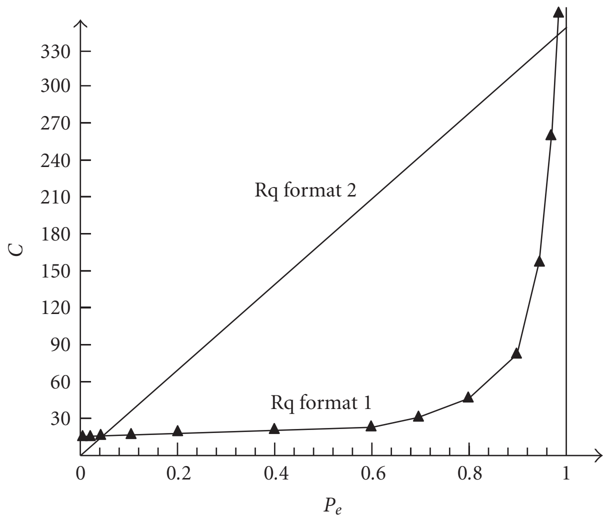

Figure 9: Estimated capacity of a request packet versus packet error rate.

which transforms into

$$C_u = (1 - P_e)C_{u-1} + P_e C_{u-2} + \frac{1}{1 - P_e}, \qquad (7)$$

with the boundary conditions $C_1 = 1$ and $C_0 = 0$.

The capacity of a format 2 request packet is more difficult to estimate (at least in all circumstances) because of the multitude of cases. Note, however, that in those scenarios when that format is expected to bring most help, the density of missing blocks will result in most of them being represented as bit maps. Then, the capacity of a format 2 packet can be simply approximated as

$$C'_u = B \times P_e, \qquad (8)$$

where B is the maximum number of bits in the packet available for the bit map. This approximation is going to work well at least for a medium range of P_e.

Figure 9 compares the two functions for $M = 16$ and $B = 352$, which values match the actual parameters of request packets in our system. format 2 appears to clearly win, except for very low (less than 4%) and extremely high (larger than 97%) error rate. Note that approximation 6 ceases to work in those regions. For the low end, that happens when less than one of every 40 blocks is missing (around $P_e = 0.025$), in which case skips will prevail over maps (see Section 3.6), but we can confidently say that below this “phase transition” threshold format 2 is going to be slightly worse than format 1, because its representation of individual blocks is less efficient (4 bytes per block instead of 3). Needless to say, the other extreme (error rates approaching 1) is irrelevant. For P_e between 0.05 and 0.9, the simple formula (8) gives in fact a very good approximation, at least as long as packet errors are independent.

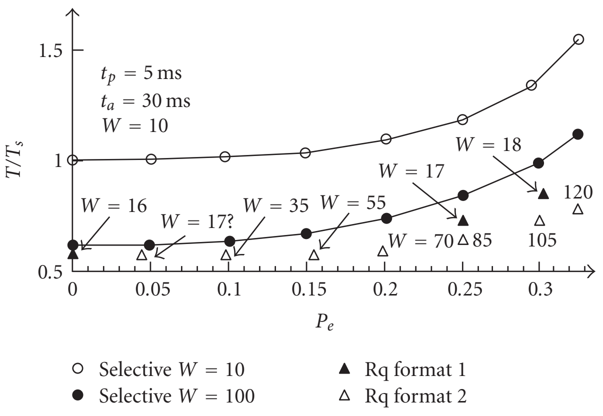

Figure 10: Our scheme versus selective acknowledgments with fixed window size (analytical comparison).

Using the values produced by formulas (7) and (8), one can toy with (1) and (4). One simple way to tweak that model to describe our scheme is to set in (1)

$$W_c = \begin{cases} N & \text{if } N = N_0, \\ C_u (\text{or } C'_u) & \text{otherwise,} \end{cases} \qquad (9)$$

where N_0 is the total number of blocks in the transmitted take. As formula (4) requires W_c to be an integer number, we can restrict the application of (1) to those values of P_e that generate integer values of C_u (or C'_u) and interpolate for other values.

Figure 10 shows the result of applying the new model to the case of transmitting 10 000 blocks. The two variants of our scheme are compared to two instances of the straightforward selective acknowledgment protocol with a fixed window size. Our protocols are represented by discrete points, to emphasize the fact that only certain values of P_e can be handled by the model. In particular, the expected window size grows rather slowly for format 1, and only three integer values (16, 17, 18) show up for $P_e < 0.4$. The value for $P_e \approx 0.048$ (window size 17) for format 2 has a question mark, as the point is close to the phase transition threshold and thus not reliable. Note that a standard selective acknowledgment scheme with fixed window size is bound to lose slightly, even when the error rate is zero, because of the need to interrupt transmission at the window boundary and momentarily reverse the channel.

4. Empirical Verification

4.1. Packet Losses. In a real-life deployment of an HDL-CPP system, one can distinguish two scenarios when a packet can be lost. First, even with the absence of explicit external interference (e.g., from another RF device), a packet can be lost “statistically” because of the background noise. Under normal operating conditions, which assume the maximum transmission range of 30 meters and the lack of interference from another simultaneous transmission (involving a different HDL), the rate of such errors is below

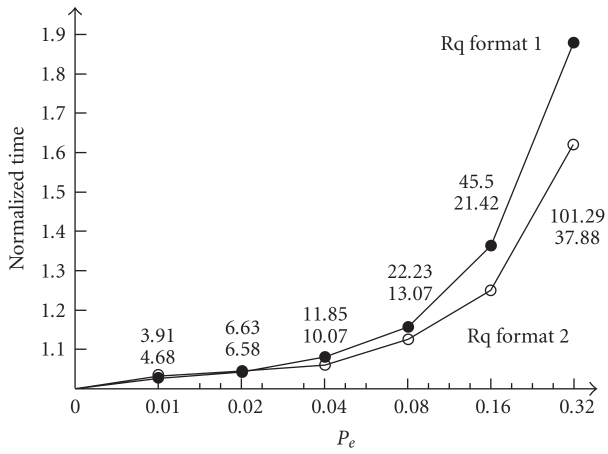

FIGURE 11: Normalized duration of transfer versus packet error rate under "random loss" conditions.

1% and they appear to be random and truly independent events.

The second scenario type involves an interference from other HDL/CPP setups operating nearby on the same channel. Losses in such a case can be longer and correlated, namely, runs of missed packets are more likely. While such situations are not representative of normal operation (and they are avoidable by the application, see Section 5.2), we also studied them to assess the resilience of our system to explicit RF interference causing losses in excess of 50% of packets.

4.2. Observed Performance. Table 1 shows the distribution of error runs under the "random" losses. By intentionally crippling the system, that is, attenuating the received signal level beyond normal operating conditions, we pushed the packet error rate over 30%. The attenuation was accomplished by reducing the transmission power, trimming the antenna, and/or moving the two nodes (HDL and CPP) further apart until the average observed packet error rate matched the first column of Table 1. The remaining columns show the fraction of all observed error runs (a consecutive sequence of erroneous packets was treated as a single sample in these statistics). As we can see, there are no long series of lost packets, which means that attempts at identifying runs (that would reduce the number of retransmission requests) will not be very successful.

Figure 11 shows the increase in the time of take transfer depending on the packet error rate under "random loss" conditions. The measured value is the ratio of the actual transfer time to the time with zero losses. The two sets of points correspond to the two request formats described in Section 3.6. At each point we also show the average number of rounds taken by each format (the upper number is format 1) for a 60-second (4000-block) take.

Note that for a low error rate (below 4%), format 1 turns out to be marginally better than format 2, which trend significantly reverses for higher error rates. This is explained by the slightly less efficient representation of sparse missing blocks by format 2 under very low error rates (see Section 3.7).

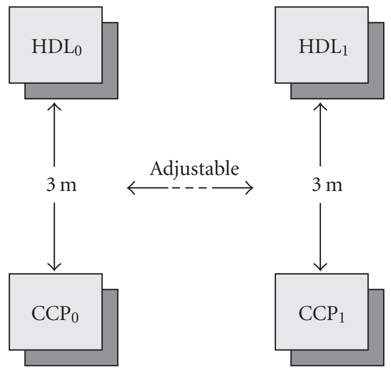

FIGURE 12: An adjustable configuration of two interfering systems.

Scenarios with short RF interferences, for example, corresponding to the situation when one HDL is sending blocks, while some others exchange status messages with their CPP's, are not visibly different from the random scenario, with the appropriately adjusted packet error rate. Consequently, it is more interesting to look at the cases of large losses caused by two or more concurrent take transfers. By adjusting the distance (cross-interference) between the different setups, and looking at the behavior of a single HDL-CPP pair, we can obtain error rates above 50% and reaching 90–95%. Note that even in extreme interference scenarios, the loss is never 100%, which is mostly due to the capture effect [33, 34]. What we see at those higher loss rates is longer error runs.

Experiments illustrating the performance of our system under such large loss rates were carried out using the setup shown in Figure 12. The two HDL-CPP pairs were constantly transmitting long takes in parallel. The separation between the two pairs was adjusted (between 0.5 and 10 m) to match the prescribed average error rate.

Figure 13 presents the observed distribution of runs for three different average packet error rates. The length of each bar tells the percentage of all erroneous (lost) packets that belonged to runs of the corresponding length (marked on the x-axis). Runs of length 1 are not shown (their percentage can be trivially deduced as the difference between 100 and the length of the first bar). In particular, for the packet error rate of 0.9, 50% of all lost packets were lost in runs of 17 or more.

The actual performance of an HDL-CPP pair under heavy interference conditions is illustrated in Figures 14 and 15. These results were obtained by making one HDL of the setup in Figure 12 transmit a continuous sequence of samples (belonging to a dummy infinite take), while the other pair tried to exchange a sequence of forty 60-second takes. (The CPP of that pair was irrelevant and completely inactive.) For each transfer, we measured the total transmission time (until the arrival of the last missing sample) as well as the number of rounds needed to accomplish the transfer (the first transmission in response to the initial request was counted as round 1). Both measures were averaged over the 40 experiments. As before, the average transmission time per take is normalized assuming that a completely error-free transmission lasts one unit.

Table 1: Percentage of error runs under random loss.

PER	RL = 1	RL = 2	RL = 3	RL = 4	RL = 5	RL = 6	RL = 7	RL = 8
0.005	0.9814	0.0173	0.0013	—	—	—	—	—
0.010	0.9591	0.0398	0.0012	—	—	—	—	—
0.020	0.9253	0.0686	0.0058	0.0003	—	—	—	—
0.040	0.8553	0.1263	0.0166	0.0017	—	—	—	—
0.080	0.7401	0.1954	0.0498	0.0108	0.0028	0.0006	0.0004	—
0.160	0.5945	0.2273	0.1035	0.0420	0.0181	0.0085	0.0033	0.0016
0.320	0.5173	0.2484	0.1257	0.0593	0.0246	0.0135	0.0062	0.0029

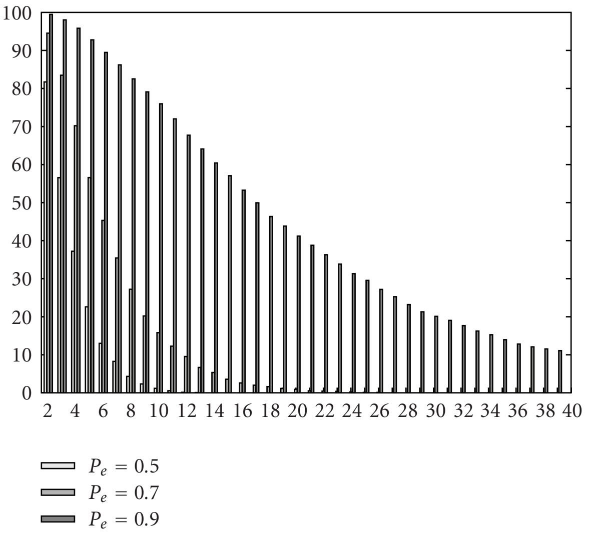

Figure 13: Error run length distribution under heavy losses incurred by interference.

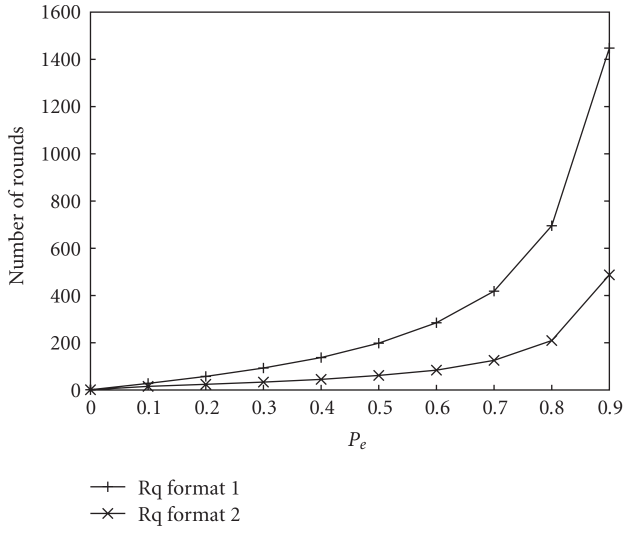

Figure 14: The average number of rounds to transmit a 60-second take under heavy interference.

In terms of rounds, the superiority of the second request format is clear. Note that under enormously large error rates, a fewer number of rounds requires fewer request packets, which begins to positively feed back into the overall transmission time (the impact of losses among the request packets is smaller). Naturally, the actual transmission time of a take grows quite significantly with the increased interference level. Intuitively, with two independent pairs operating in parallel, one would be willing to put up with a twofold increase in the average take transmission time. Based on Figure 15, this happens around $P_e = 0.4$, which was observed for the separation distance of about 5 m.

5. Enhancements and Generalizations

Although the solution discussed in this paper has been devised as part of a very practical project aimed at the development of a specific device catering to a specific application, it addresses a general problem that may be of relevance in other applications. In particular, we are currently building a wireless device for a profile matching application where the role of *takes* from the HDL is played by small pictures (photographs) exchanged by the nodes. The data exchange protocol in the new application has been copied verbatim from the HDL program.

5.1. Trading Reliability for Complexity. One of the characteristics of good "holistic" software solutions for micro-controlled devices is their breaking away from the layered paradigm of large-scale computing and networking. In consequence, some traditional concepts may assume different roles. This has happened to the concept of transmission window in our scheme. In contrast to traditional protocols, where the window counteracts the negative impact of the bandwidth-delay product, its primary purpose in our system is to reduce the channel reversal penalty (something that the traditional protocols have never worried about).

Our scheme can be extended to a scenario where the transmitted data has the appearance of a continuous stream, which can only be partially stored at the sending node. Of course, similar to other schemes [9, 28], it cannot possibly guarantee in such circumstances that all packets of the stream will always make it to the destination. But, as we argued elsewhere [35], all *true* streams must be prepared to accept occasional losses. With this allowance, it is easy to modify our solution to work with a circular buffer at the sender in a way that will automatically trade the buffer size for the perceived (effective) loss rate at the destination.

Let M be the size of the circular buffer at the sender, that is, the buffer can accommodate up to M most recent

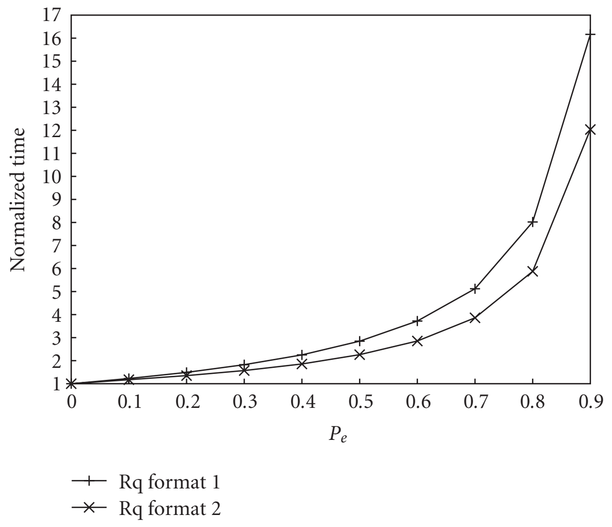

FIGURE 15: The average normalized amount of transmission time under heavy interference.

outgoing blocks of data. Let K be the number of the first block stored in the buffer. The number of the last stored block is $K+M-1$. The process responsible for generating the blocks to be sent to the other party simply fills in the buffer in the standard way discarding the oldest samples in the natural FIFO fashion.

The modified scheme operates according to the original paradigm whereby rounds are triggered by single-packet requests specifying as many missing blocks as possible. Each of the outgoing block packets includes in its header the current value of K, to tell the recipient the minimum number of block that it can still request. If a block whose number is less than K is missing, the recipient knows that it has been irretrievably lost (it makes no sense to ask for that block). Otherwise, the protocol executes exactly as before.

The block numbers can be stored in a modular fashion, such that even huge streams can be accommodated without overtaxing the short packet format. In order to know which blocks to request, the recipient must maintain a conceptual equivalent of the circular buffer, which can be reduced to a bit map, if the received blocks need not be stored at the node.

5.2. Collision Avoidance. By consistently obeying a simple set of rules, multiple nodes of the application, including multiple CPPs, as well as multiple HDLs controlled by the same CPP, can effectively avoid collisions among simultaneous take transfers and make sure that status packets do not interfere with takes. This can be accomplished in a way that properly accounts for hidden terminals.

First, consider the case of a single CPP servicing multiple HDLs. In this setup, only one take transmission can be active at any given time, as each of them must be initiated by the CPP. Consider two HDL nodes A and B. Suppose that A is sending a round of blocks to the CPP. If B is located within the transmission range of A, it will refrain from sending a status packet for as long as A is transmitting. This is because status packets are sent with LBT enabled, which means that B will listen to the medium for a short while before transmitting and postpone its transmission when it senses another activity. To make this work, one has to make sure that the LBT interval is longer than the tiny interpacket gap separating two packets in a round. This is analogous to SIFS/DIFS spacing in 802.11 [21, 22].

In a sense, the back-to-back series of packets sent by the HDL in one round can be viewed as a single unit of transmission. The situation is in fact much more favorable, because even if B damages the first block of the round (LBT race), it will necessarily yield to the remaining packets. Thus, the first block packet sent in a round can be viewed as a bandwidth reservation request addressed to all HDL nodes in A 's neighborhood. One may even consider putting a special (irrelevant) packet in front of a round batch, to make sure that LBT races never damage actual block packets.

Now suppose that B is out of A's range, but within the range of the CPP, that is, it is a hidden terminal from the viewpoint of A. Then, B has had an opportunity to see the CPP's request addressed to A. Note that that request specified (implicitly) the total number of blocks expected from A. Nothing stops B from decoding that request (the same way it would decode a similar request addressed to itself) and estimating for how long A will be transmitting the blocks. This estimate can be quite accurate, as round transmissions are highly deterministic. Consequently, B will be able to hold back its traffic until the CPP has completely received the round.

In an environment with multiple CPPs, one has to avoid a situation when two CPPs simultaneously initiate take transmissions in a way that can make them interfere. Note that the HDLs involved in those transmissions can be allowed to interfere, as long as the reception at their CPPs is clear. Let us denote the two CPPs by C_1 and C_2, with H_1 and H_2 being their respective HDLs. The problem only occurs if H_1 is within the range of C_2 or H_2 is in the range of C_1. Suppose we have the first scenario (the other is obviously symmetric). Being within the range of C_2, H_1 will have an opportunity to hear C_2's request addressed to H_2. Consequently, it will not be responding to the requests of its CPP (C_1) for the amount of time inferred from the overheard request packet. This way, H_1 will not start a round transmission for as long as H_2 is handling the request of its CPP.

Notably, the nature of traffic in our application automatically takes care of the exposed terminal problem as well. The only rule to be added to the discussed scheme is that once an HDL decides that it is safe to transmit (according to the above rule), it will respond to a round request from its CPP blindly, that is, without employing LBT for the first packet. We mention this explicitly (even though we agreed that packets within a round are to be sent without LBT), because one might be tempted to send the first packet of a round with LBT enabled (in the spirit of treating the round as a logically single activity), but if the HDL knows of no foreign CPP requests in its neighborhood, it need not be polite to the neighbors with the transmission of its round. This is because any collision of that round will be purely local; both CPPs will be able to receive their blocks.

This discussion demonstrates that the interference experiments described in Section 4.2 do not pertain to scenarios expected to haunt real deployments, but were merely aimed at assessing the performance of our application under extreme stress. On top of the explicit collision avoidance techniques, there exist other ways to separate multiple CPP-HDL setups operating in the same area. First, one can assign different channels to those setups. Formally, the CC1100 RF module offers up to 256 different channels [11], whose separation can be improved by reducing their number. When using only 16 of those channels, that is, with 16 internal channels separating two adjacent application channels, the distance separation of 1 m between the two pairs resulted in the observed error rate below 2%. Creative power management is another option. In the present application, the CPP can ask its HDLs to adjust their transmitted power levels. Note, however, that by varying power levels used by different nodes, we spoil the approximate symmetry of neighborhood perception, which in turn affects the performance of reciprocity-based collision avoidance schemes.

5.3. Other RF Modules. Our choice of CC1100 for the project was dictated by its reasonable friendliness in terms of program interface as well as simplicity in terms of the built-in functionality. The module is a rather typical representative of its class. In particular, other variants of the same line by Texas Instruments, including CC2400, CC2420, and CC2430, offer essentially the same basic functionality augmented by on-chip MAC/routing features aimed at ZigBee compliance. Note that those modules operate in the 2.4 GHz band, which was considered unacceptable for the application for formal reasons. Even if the RF frequency was not an issue, the extra features of those modules would be useless for the application; consequently, they would essentially operate in a CC1100-compatible mode.

Note that the efficiency of take transmissions in our project hinges on exploiting the largest possible fraction of the raw bandwidth offered by the RF module. Thus, any advanced built-in features aimed at collision avoidance, bandwidth policing, and so on, would only get in the way. This property of our solution essentially puts all RF modules into the same basket.

In some of our other projects we have been using even simpler RF modules, for example, TR8100 form RF Monolithics [36], which, despite drastic differences in the interface, offers essentially the same functionality as CC1100. The primary difference is the absence of an on-chip LBT mechanism in TR8100, which, however, as we have verified elsewhere [8], can be effectively and efficiently emulated in software. Consequently, one should expect about the same results with other RF devices, as long as they offer similar transmission rates, which is the dominating factor affecting the overall performance of our scheme.

6. Conclusions

We have presented and analyzed a simple protocol for reliably transmitting file-like chunks of data between low-cost wireless devices. Our objective, dictated by the constraints of the application for which the protocol was specifically designed, was to maximize the bandwidth available to such transfers. The primary factor making the problem different from its classical formulation was the simplicity of the wireless channel whose raw capacity was tightly matched to the transmission bandwidth required by the application. By identifying and understanding the limitations of the platform, we were able to accomplish our goal and make the best use of its small (albeit sufficient) resources. Despite its stimulation by a very specific application, the problem appears to be quite general; our solution can be used to transmit reliably any file-like objects.

As a side effect of our presentation, we have demonstrated how good solutions in the embedded world can be arrived at by employing a holistic approach to the problems. Our data transmission scheme is a holistic derivative of some exotic properties of the RF module, flash memory, the limited amount of RAM, as well as the application-level demands. On the one hand, one may feel disappointed by this interference of the various apparently unrelated "features" into something that should rightfully belong to a well-established and separated "layer." On the other hand, it is reassuring to see this much potential for creativity in an otherwise routine project. This potential is what makes the realm of embedded systems challenging in its own highly attractive sort of way.

References

[1] B. M. Baker, "Ballistocardiography: predictor of coronary heart disease," *Circulation*, vol. 37, no. 1, pp. 1–3, 1968.

[2] O. Such, J. Muchlsteff, R. Pinter, et al., "On-body sensors for personal healthcare," in *Advances in Healthcare Technology: Shaping the Future of Medical Care*, G. Spekowius and T. Wendler, Eds., vol. 6, pp. 436–488, Springer, Berlin, Germany, 2006.

[3] S. Lin, D. J. Costello Jr., and M. Miller, "Automatic-repeat-request error-control schemes," *IEEE Communications Magazine*, vol. 22, no. 12, pp. 5–17, 1984.

[4] R. van Renesse, "Masking the overhead of protocol layering," in *Proceedings of the ACM SIGCOMM Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, vol. 26, pp. 96–104, Palo Alto, Calif, USA, August 1996.

[5] L. Song and D. Hatzinakos, "A cross-layer architecture of wireless sensor networks for target tracking," *IEEE/ACM Transactions on Networking*, vol. 15, no. 1, pp. 145–158, 2007.

[6] R. Jurdak, *Wireless Ad Hoc and Sensor Networks: A Cross-Layer Design Perspective*, Springer, Berlin, Germany, 2007.

[7] M. B. Abbott and L. L. Peterson, "Increasing network throughput by integrating protocol layers," *IEEE/ACM Transactions on Networking*, vol. 1, no. 5, pp. 600–610, 1993.

[8] P. Gburzynski and W. Olesinski, "On a practical approach to low-cost ad hoc wireless networking," *Journal of Telecommunications and Information Technology*, vol. 2008, no. 1, pp. 29–42, 2008.

[9] Q. Liu, S. Zhou, and G. B. Giannakis, "Cross-layer combining of adaptive modulation and Coding with truncated ARQ over wireless links," *IEEE Transactions on Wireless Communications*, vol. 3, no. 5, pp. 1746–1755, 2004.

[10] C. Nagy, *Embedded Systems Design Using the TI MSP430 Series*, Elsevier, Amsterdam, The Netherlands, 2003.

[11] Texas Instruments, "CC1100 Single Chip Low Cost Low Power RF Transceiver," 2006, http://focus.ti.com/lit/ds/symlink/cc1100.pdf.

[12] E. Akhmetshina, P. Gburzynski, and F. Vizeacoumar, "PicOS: a tiny operating system for extremely small embedded platforms," in *Proceedings of the International Conference on Embedded Systems and Applications (ESA '03)*, pp. 116–122, Las Vegas, Nev, USA, June 2003.

[13] W. Dobosiewicz and P. Gburzynski, "From simulation to execution: on a certain programming paradigm for reactive systems," in *Proceedings of the 1st International Multiconference on Computer Science and Information Technology (FIMCSIT '06)*, pp. 561–568, Wisla, Poland, November 2006.

[14] P. Gburzynski, B. Kaminska, and W. Olesinski, "A tiny and efficient wireless ad-hoc protocol for low-cost sensor networks," in *Proceedings of the Conference on Design, Automation and Test in Europe (DATE '07)*, pp. 1557–1562, Nice, France, April 2007.

[15] W. Lynch, "Computer systems: reliable full-duplex file transmission over half-duplex telephone line," *Communications of the ACM*, vol. 11, no. 6, pp. 407–410, 1968.

[16] K. A. Bartlett, R. A. Scantlebury, and P. T. Wilkinson, "A note on reliable full-duplex transmission over half-duplex links," *Communications of the ACM*, vol. 12, no. 5, pp. 260–261, 1969.

[17] C. Barakat, E. Altman, and W. Dabbous, "On TCP performance in a heterogeneous network: a survey," *IEEE Communications Magazine*, vol. 38, no. 1, pp. 40–46, 2000.

[18] H. O. Burton and D. D. Sullivan, "Errors and error control," *Proceedings of the IEEE*, vol. 60, no. 11, pp. 1293–1301, 1972.

[19] H. Liu, H. Ma, M. El Zarki, and S. Gupta, "Error control schemes for networks: an overview," *Mobile Networks and Applications*, vol. 2, no. 2, pp. 167–182, 1997.

[20] D. Chkliaev, J. Hooman, and E. de Vink, "Verification and improvement of the sliding window protocol," in *Proceedings of the 9th International Conference on Tools and Algorithms for the Construction and Analysis of Systems (TACAS '03)*, vol. 2619 of *Lecture Notes in Computer Science*, pp. 113–127, Springer, Warsaw, Poland, April 2003.

[21] B. P. Crow, I. Widjaja, L. G. Kim, and P. T. Sakai, "IEEE 802.11 wireless local area networks," *IEEE Communications Magazine*, vol. 35, no. 9, pp. 116–126, 1997.

[22] X. Ling, L. X. Cai, J. W. Mark, and X. Shen, "Performance analysis of IEEE 802.11 DCF with heterogeneous traffic," in *Proceedings of the 4th Annual IEEE Consumer Communications and Networking Conference (CCNC '07)*, pp. 49–53, Las Vegas, Nev, USA, January 2007.

[23] S. Xu and T. Saadawi, "Does the IEEE 802.11 MAC protocol work well in multihop wireless ad hoc networks?" *IEEE Communications Magazine*, vol. 39, no. 6, pp. 130–137, 2001.

[24] G. Bianchi, "Performance analysis of the IEEE 802.11 distributed coordination function," *IEEE Journal on Selected Areas in Communications*, vol. 18, no. 3, pp. 535–547, 2000.

[25] A. Rahman and P. Gburzynski, "Hidden problems with the hidden node problem," in *Proceedings of the 23rd Biennial Symposium on Communications*, pp. 270–273, Kingston, Canada, May-June 2006.

[26] J. F. Kurose and K. W. Ross, *Computer Networking: A Top-Down Approach Featuring the Internet*, Addison-Wesley, Reading, Mass, USA, 2004.

[27] L. Libman and A. Orda, "Optimal sliding-window strategies in networks with long round-trip delays," *Computer Networks*, vol. 46, no. 2, pp. 219–235, 2004.

[28] M. C. Chan and R. Ramjee, "Improving TCP/IP performance over third-generation wireless networks," *IEEE Transactions on Mobile Computing*, vol. 7, no. 4, pp. 430–443, 2008.

[29] B. Scheuermann, C. Lochert, and M. Mauve, "Implicit hop-by-hop congestion control in wireless multihop networks," *Ad Hoc Networks*, vol. 6, no. 2, pp. 260–286, 2008.

[30] J. Chen, M. Gerla, Y. Z. Lee, and M. Y. Sanadidi, "TCP with delayed ack for wireless networks," *Ad Hoc Networks*, vol. 6, no. 7, pp. 1098–1116, 2008.

[31] Q. Zhang and S. A. Kassam, "Hybrid ARQ with selective combining for fading channels," *IEEE Journal on Selected Areas in Communications*, vol. 17, no. 5, pp. 867–880, 1999.

[32] E. Malkamaki and H. Leib, "Performance of truncated type-II hybrid ARQ schemes with noisy feedback over block fading channels," *IEEE Transactions on Communications*, vol. 48, no. 9, pp. 1477–1487, 2000.

[33] I. Demirkol, C. Ersoy, and F. Alagoz, "MAC protocols for wireless sensor networks: a survey," *IEEE Communications Magazine*, vol. 44, no. 4, pp. 115–121, 2006.

[34] K. Whitehouse, A. Woo, F. Jiang, J. Polastre, and D. Culler, "Exploiting the capture effect for collision detection and recovery," in *Proceedings of the 2nd IEEE Workshop on Embedded Networked Sensors (EmNetS '05)*, pp. 45–52, Sydney, Australia, May 2005.

[35] C. Baransel, W. Dobosiewicz, and P. Gburzynski, "Routing in multihop packet switching networks: Gb/s challenge," *IEEE Network*, vol. 9, no. 3, pp. 38–61, 1995.

[36] R. F. Monolithics, "TR8100 916.5 MHz Hybrid Transceiver," 1999, http://www.rfm.com/products/data/tr8100.pdf.