Reliable Data Transmission Over Simple Wireless Channels: a Case Study

Pawel Gburzynski
Department of Computing Science
University of Alberta
Edmonton, Alberta, Canada
pawel@cs.ualberta.ca

Bozena Kaminska
School of Engineering Science
Simon Fraser University
Burnaby, BC, Canada
kaminska@sfu.ca

Ashikur Rahman
School of Engineering Science
Simon Fraser University
Burnaby, BC, Canada
arahman@sfu.ca

Abstract

We discuss the issue of reliable transmission of bounded streams of data samples over simple wireless channels offered by low-end RF modules. Traditional solutions, i.e., window-based ARQ schemes, applied to such channels tend to waste bandwidth, because of the need to maintain duplex communication. Our case study shows how one can overcome the inherent limitations of low-bandwidth unstructured wireless channels and claim most of their capacity for transmitting useful data.

Keywords: embedded systems, wireless communication, reliability

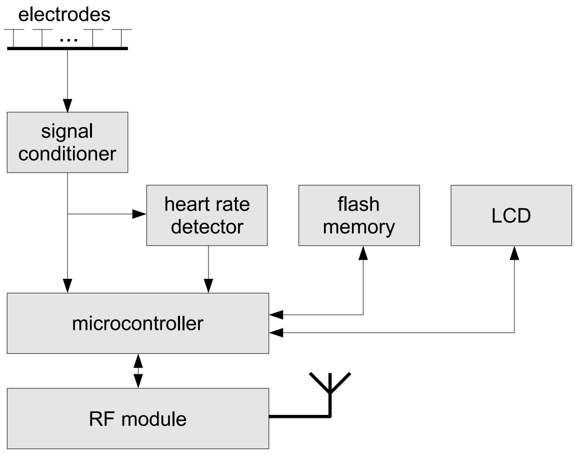

Figure 1. Block diagram of the Heart Data Logger (HDL).

1. Introduction

The data transmission problem discussed here arose as part ot the design of a wireless sensing device for heart monitoring based on ballistocardiography [13]. The primary function of the device, dubbed HDL (Heart Data Logger) in the sequel (see Figure 1), was to collect data samples from a set of accelerometers, store them temporarily in local flash memory, and transmit them reliably (in near real time) to a Collection and Processing Point (CPP). Cost considerations combined with restrictions regarding the RF bandwidth, and 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 [8] brought the system down to its knees. 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 studied alternatives to those obvious and popular schemes and were able to arrive at a solution that met our objectives.

In addition to being a case study, whose conclusions, in our opinion, can be extrapolated on many other wireless sensing applications, our work demonstrates that the realm of small embedded wireless systems brings about a number of idiosyncrasies that often force one 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 [15, 12, 5, 1]. One has to resort to cases of product engineering to show those issues in the proper light of their highly practical relevance.

2. The problem

The essential hardware components of the HDL are the MSP430F1611 microcontroller [10] and the CC1100 RF module [14] (both from Texas Instruments). The sensor data input to the HDL consists of eight analog signals arriving from the accelerometers placed on the subject's body. Those signals are fed to ADC ports of the microcontroller and converted into 48 bytes of data (called a sample) at the rate of 125 samples per second, yielding 48,000 bps of the incoming data bandwidth.

The ADC converter, as well as the sample collection process, are turned on upon request from the CPP. The prescribed number of samples (called a *take*) are then collected and stored in flash memory at the HDL. They can be transmitted in parallel with their collection, or merely collected and stored locally with the intention of being retrieved later. One of the key requirements was a near real time transmission of the takes to the CPP. This means that the amount of time needed to transmit a take should not significantly exceed the time of its acquisition.

The flash memory plays a dual role. From the viewpoint of transmitting BCG samples to the CPP, it acts as a buffer compensating for transmitter's jitter and allowing the node to retransmit missed samples. It also acts as a simple database storing the last few sample streams taken from the subject, which can be transmitted retroactively upon request from the CPP. That functionally was explicitly requisitioned as a means of auditing the sampled data and providing for an extra degree of reliability. Even with a total blackout regarding the communication with its CPP, the HDL is still able to complete collecting the last-requested sample.

The CPP is built around a standard PC (running Linux) with an RF Access Points (AP) device interfaced to it via USB. The AP looks like a simplified HDL: it has no accelerometer connector, no signal decoding modules, no flash memory, and no LCD. Instead, it is equipped with a USB interface which emulates RS232 over USB at the rate of $115,200$ kbps. The functionality of AP consists in forwarding data packets between the PC and the population of bound HDL nodes.

The communication protocol between the HDL and its CPP consists of several simple commands and responses, which, except for the take transmission, require a trivially small amount of bandwidth. Standard solutions to the problem of reliable data transmission over unreliable channels are known under the generic name of ARQ protocols [8, 2]. 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 [3]. Some schemes employ negative acknowledgments [9, 4] to explicitly describe what has been missing, some others relay solely on timeouts.

The most difficult problem facing a sensible implementation of 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 a reliable and deterministic isolation of the feedback channel. The range of transmission rates available to CC1100 is adjustable within a certain interval, with the raw limit around 200 kbps, which, considering the coding (Manchester) and packetization reduces to about 50 kbps of effective bandwidth. Specifically, under favorable conditions, the HDL transmitter is able to send 56-byte packets (needed to accommodate samples) back-to-back at 6-7ms intervals. Note that this is only marginally above the data acquisition rate ($48,000$ bps), which means that the device is practically stressed to its limits.

CC1100 (which is a good representative of RF devices in its class) comes equipped with rudimentary tools for collision avoidance [14]. 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. While the logistics of the CPP station precludes receiving more than one take at the same time, it admits simultaneous supervision of multiple HDLs. Note that an HDL is able to collect a take without transmitting it immediately to the CPP.

Consequently, our driver of the RF module implements a simple LBT (Listen Before Transmit) scheme, whereby a node perceiving a radio activity before transmission backs off to avoid collision at its end. Such a scheme greatly facilitates low-bandwidth communication among multiple nodes, but it also taps into the bandwidth, some of it being wasted on the back-offs. Note that all request and status messages, which we would like to be able to coexist, are extremely short. Thus, complex handshakes of the RTS-CTS-DATA-ACK variety are useless (and would be harmful [11]) under the circumstances.

Under ideal conditions, i.e., no interference from other nodes, the HDL transmitter is able to send packets back-to-back reasonably fast, with minimal inter-packet-spacing. However, as soon as the collision avoidance mechanism kicks in (i.e., foreign activity is sensed and back-off 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. Practically, the smallest delay incurred by an attempt to receive a packet while transmitting (a channel reversal) is about 30ms. Complex handshakes of the RTS-CTS-DATA-ACK variety are out of the question [11].

3. Performance of traditional ARQ schemes

The first protocol tried in our design operated as follows. When the CPP wants to retrieve a take from the HDL, it sends a short SEND request specifying two parameters: F–the starting sample number, and N–the number of samples to be retrieved. Having received the initial SEND request, the HDL starts to transmit the requested samples consecutively. As the sample number is included in every packet, the receiving node can tell which samples have made it and which have been lost.

To avoid too many acknowledgments, the HDL maintains two counters: *NextToGo* and *NextToAck*. The first counter tells the number of the next sample to be transmitted, while the second one points to the first sample for which an acknowledgment (meaning a SEND request) has not been received yet. The device is allowed to keep sending samples until *NextToGo* $-$ *NextToAck* $< W$, where W is the assumed window size. To facilitate this operation, the CPP will refrain from acknowledging every single sample. Ideally, it would like to get away with a single acknowledgment per the entire window.

Having reached the end of a window, for as long as *NextToAck* is not advanced, the HDL keeps retransmitting the last sample. If the CPP has missed some packets from the window, it should repeat the last SEND message until the missed fragment arrives. A duplicate SEND request for the *NextToAck* sample 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 filled 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 [6]. The key to its effectiveness in the present application is in the right selection of the window size W, the intervals between SEND messages and timeouts. Most of all, 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 usually 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 (which tends to reduce the maximum effective throughput) by turning the channel into a pipe [7]. In our case, the propagation delay is insignificant, and the window is used as the grain of acknowledgment—to avoid too many costly channel reversals.

Most of the insight into the problem can be obtained from a very simple model. Let W denote the window size, t_p be the time required to transmit a single sample packet, t_a represent the cost of receiving an acknowledgment, and P_e stand for the packet error probability. With sparse acknowledgments ideally separated as widely as the window size, the approximate amount of time needed to transmit a take of N samples can be expressed as:

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

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

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

is the probability that exactly n initial samples of m total have been transmitted successfully.

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 sent only 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 reception opportunities) avoiding much of the overhead. The

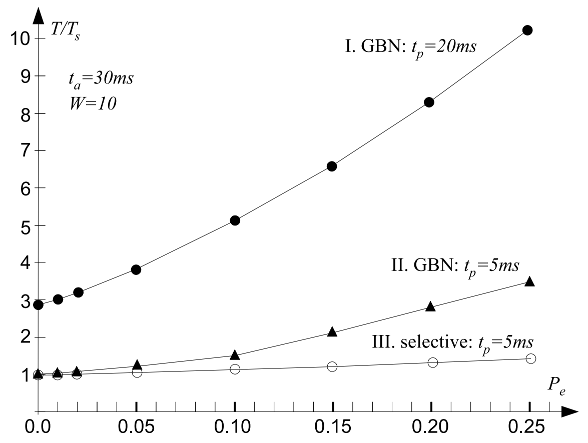

Figure 2. Normalized time of sending a series of samples under Go-Back-N and selective schemes.

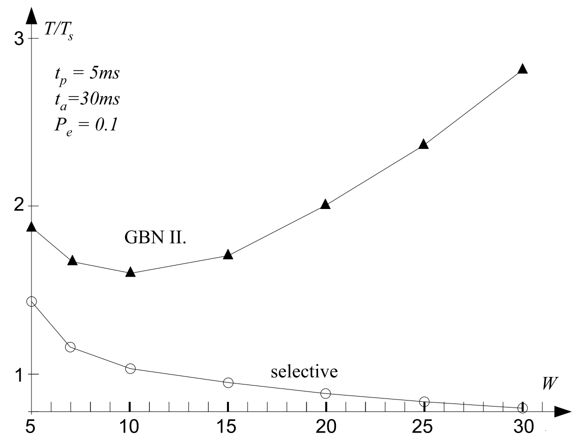

Figure 3. Behavior of Go-Back-N and selective schemes for different window size.

advantage is illustrated in Figure 2, which compares the normalized time of transmitting a long series of samples 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 actual transmission time T to the time T_s it would take to send the samples back-to-back with no errors and no reception opportunities anywhere within the sequence. T_s practically corresponds to the take acquisition time, which is also the minimum amount of time that the CPP can ever expect. The window size was 10 samples. In the first (spontaneous) case, t_p is set to 20ms, 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 30ms 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.

Once we conclude that acknowledgments should only be conveyed 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 samples that were missing in the window. Then, the next window would begin with the missing samples from the previous one. The performance of this new scheme is captured by Equation 1 with the probability $P(n,m)$ redefined as:

$$P(n,m) = \binom{m}{n} \times (1 - P_e)^n \times P_e^{(m-n)} \tag{3}$$

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

The solution for $W_c = 10$ is depicted by curve III in Figure 2. Its superiority over the Go-Back-N approach is clear. One of its desirable features is strict monotonicity with respect to the window size (see Figure 3), which is not the case with the Go-Back-N scheme (which tends to retransmit more and more redundant packets as the window becomes large).

4. The workable scheme

The monotonicity of the simple selective scheme with respect to the window size W suggests that W should be as large as possible. Note, however, that one would like to avoid a situation when the feedback message itself must consist of multiple packets, because then we would have to cope with the issue of reliable reception of the feedback packet itself. In a situation when the feedback message consists of multiple packets, simple persistent and idempotent schemes will not work, which will bring about more wastage of the precious bandwidth on multiple channel reversals. Consequently, we adopt the following set of rules:

1. The window size is determined by the (maximum) number of requested samples whose description can fit into a single feedback packet.

2. The end of a window is explicitly indicated by the HDL in a persistent manner until noticed by the CP.

3. Feedback 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 samples and then expects to receive them in the next batch of packets, doing so until all the samples have made it. At every step, the CPP asks for the maximum number of samples 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 samples to be sent and send them blindly back-to-back (no reception opportunities) until the last requested sample has been expedited. Then proceed at 3.*

3. *Send periodically, at reasonably sparse intervals (with reception opportunities enabled), a short packet indicating the end of window. Keep doing so until a new request arrives from the CPP. Then proceed at 4.*

4. *If the request is NULL, i.e., no more samples are needed, terminate the operation and proceed at 1. Otherwise, proceed at 2.*

The CPP's end of the protocol looks like this:

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

2. *Find out which samples 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 samples as possible and proceed at 3.*

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

4. *Keep receiving the samples and storing them until you see the end-of-window packet. Then proceed at 2.*

As all packets within the large and variable-size window are transmitted back-to-back (at the limit of the physical capacity of the device), the key to maximizing the performance of this scheme is the minimization of the number of rounds. This in turn directly depends on the number of feedback (request) packets needed to describe packet losses. We have implemented two formats of such packets.

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

1. Set $i = 0$.

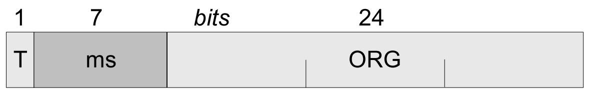

Figure 4. The layout of an element header.

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

3. If $i = n$ or $c_i > c_a$, send sample c_a. Otherwise, set $c_b = c_i$ and $i = i + 1$. Send the samples $c_b, ..., c_a$. Continue at 2.

Thus, for as long as the sample numbers are increasing, they describe individual and independent samples, while a decreasing number together with its preceding number are taken together as the boundary of a continuous range (chunk) of samples. Note that the pair $N - 1, 0$ (where N is the total number of samples 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 samples is described by a sequence of *elements*, which may identify continuous ranges of samples, 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 sample number, as shown in Figure 4. 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 sample number *ORG* to be sent to the CPP, and also sets the current location within the take to the sample number *ORG* $+ 1$. Note that the relevance of this setting is limited to the current single request packet. If *ms* is nonzero, then *ms* consecutive bytes following *ORG* are interpreted as a bit map requesting selected samples from the take fragment starting at sample number *ORG* $+ 1$. This is the sample number referred to by the first bit in the map.

The second element type (chunk) describes a continuous range of samples starting at the current position, as determined by the previous sequence of elements within the current packet. In that case, *ORG* represents the number of samples falling into the chunk. If the chunk element is the first element of the request, i.e., the current position has not been explicitly defined, it is assumed to be zero. Thus, the complete take (all samples) are described by a single element whose first byte is 0x80 and the following three bytes contain the total number of samples 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 samples immediately following the chunk. Figure 5 shows a sample request fragment, which calls for

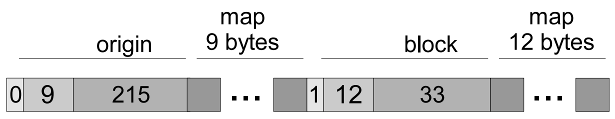

Figure 5. A request fragment.

sample number 215, then uses a bit map to select among samples 216-287. Note that one byte of the map covers 8 consecutive samples; thus, the 9 map bytes describe 72 samples starting at sample 216. The subsequent chunk element requests the continuous range of samples 288–322 (33 samples total). Finally, the second map applies to samples 323–418. The total length of the request fragment shown in Figure 5 is 29 bytes.

4.1 Types of packet losses

One can distinguish three types of packet losses that occur during sampling and simultaneous transmission:

1. Losses caused by the transmission lag. This kind of loss occurs when the transmission of samples lags behind the (hard-conditioned) sample collection process.
2. Occasional RF losses occurring during "normal" transmission. This environment corresponds to the target operating mode.
3. Heavy RF losses caused by a poor channel (long distance, interference from other RF devices). This environment represents extreme operating conditions at which the device should try to "do its best."

The first type of loss results from the fact that, besides the flash memory, there is a limited amount of storage to accommodate those samples whose transmission cannot catch up with the acquisition (which cannot be slowed down). Note that the way flash memory is organized precludes a solution with two pointers, whereby the sampling thread would write the consecutively acquired chunks at one pointer, while the sending thread would read them at a different pointer (at its own, possibly slightly slower, pace). Although possible in principle, the constant switching between reading and writing would drastically increase the average write access time, making the original problem even more serious. The problem, however, was easily circumvented by using a moderate amount of RAM (about 3KB) partitioned between 1) a small buffer for outgoing chunks awaiting transmission, and 2) a compressed list identifying the numbers of those chunks that could not be accommodated in the buffer and were not transmitted at their acquisition. At the end of the sampling stage, the HDL consults the list and transmits the missing fragments from the flash, without waiting for a series of requests from the CPP to signal their absence. This way, at the end of the first window, the HDL has transmitted all the collected samples, albeit at a slight lag compared to the sampling time (5-10%).

With this solution, the difference between an extraction of a take already stored in the HDL's database and a simultaneous reception of a take just being collected was practically eliminated. One could still see slightly higher (apparent) RF losses when the chunks were sent in parallel with their acquisition, which probably resulted from the less "clean" context of the transmitter caused by the simultaneous operation of the ADC and flash memory. As those components utilize internal oscillators, their activities are not unlikely to negatively affect the operation of the RF module.

Among the losses actually affecting some packets that were transmitted but not received we can talk about two scenarios. 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 1% and they appear to be truly random.

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 not intended for normal operation, we also studied such cases to assess the resilience of our system to explicit RF interference causing losses in excess of 50% of packets.

4.2. Observed performance

We have measured the performance of our protocol in two generic scenarios. First, we considered "normal" losses resulting from the inherent unreliability of the radio channel, which gets worse as the signal between the CPP and the HDL becomes weaker. Under the operating conditions considered normal, i.e., in the absence of explicit interference from other HDL–CPP pairs in the neighborhood, and at the separation distance below 30 meters, such losses affect 1-10% of packets. By moving the devices further apart and/or trimming their antennas, we were able to push the error rate beyond 10%. In our experiments, we first tried to reach the given target error rate, and then carried out a series of take transfers to measure the number of rounds and delays.

Next, we looked at scenarios involving adjustable interference from another HDL–CPP pair operating in the neighborhood. The experimental setup is shown in Figure 6. The two HDL–CPP pairs were constantly transmitting long

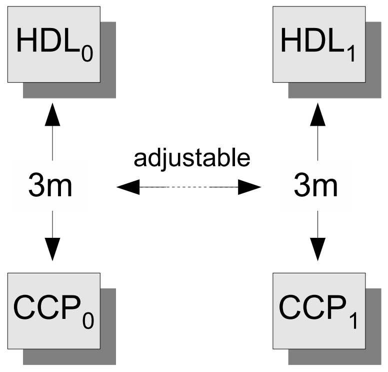

Figure 6. An adjustable configuration of two interfering systems.

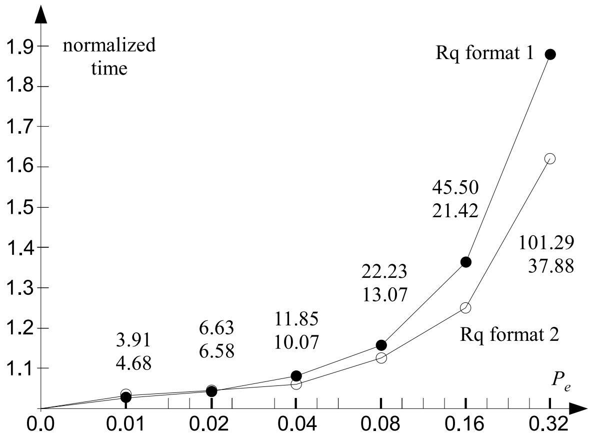

Figure 7. Normalized duration of transfer versus packet error rate under "random loss" conditions.

takes in parallel. The separation between the two pairs was adjusted (between 0.5 and 10m) to match the target average error rate, which was up to 95% in this case.

Figure 7 shows the observed increase in the time of take transfer depending on the packet error rate under "moderate 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. At each point we also show the average number of rounds taken by each format (the upper number is format 1) for a 20 second take.

Note that for a low error rate (below 4%), format 1 turns out to be slightly better than format 2, which trend significantly reverses for higher error rates. This is explained by the fact that format 1 is more compact than format 2

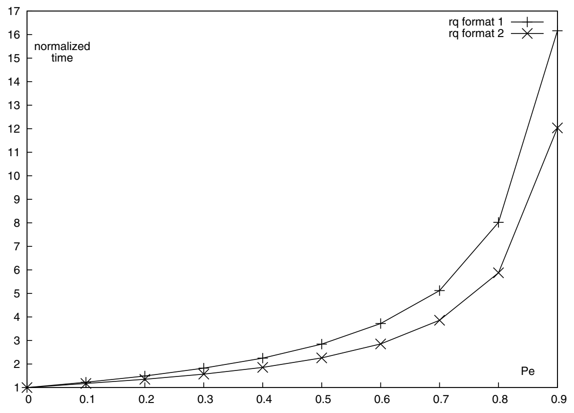

Figure 8. The average normalized transmission time under heavy interference.

for describing very sparse random missing samples. Supposing that the missing samples are so scattered (isolated) that they form no ranges or maps, format 1 accommodates $48/3 = 16$ sample numbers in a single request, while format 2 has only room for $48/4 = 12$ of them. On the other hand, when the number of missing samples is sufficiently large for maps to kick in, format 2 becomes more efficient. Assuming the impact of runs is still insignificant, the best that format 1 can produce is 16 samples per request packet, while a format 2 map entirely filling a request packet refers to $44 \times 8 = 352$ samples. Thus, if the error rate is above $16/352 \approx 4.6\%$, format 2 should start to win.

The performance of an HDL–CPP pair under heavy interference conditions is illustrated in Figure 8. These results were obtained by making one HDL of the setup in Figure 6 transmit a continuous sequence of samples (belonging to a dummy infinite take), while the other pair tried to exchange a sequence of forty 20-second takes. For each transfer, we measured the total transmission time (until the arrival of the last missing sample) averaged over the 40 experiments. As before, the transmission time is normalized assuming that a completely error-free transmission lasts one unit.

The superiority of the second request format is clear. Note that under enormously large error rates, a fewer number of rounds requires fewer requests packets, which begins to positively feed back into the overall transmission time (the impact of losses among the request packets is smaller). 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 8, this happens around $P_e = 0.4$, which was observed for the separation distance of about 5m. One should realize that there exist at least two easy ways of reducing the interference of multiple pairs operating in the same neighborhood: using different channels and/or adjusting the transmission power. Formally, the CC1100 RF module offers up to 256 different channels [14], whose separation can be improved by reducing their number. At $1/16$ of the maximum channel separation (i.e., with 16 channels instead of 256), the distance separation of 1m between the two pairs resulted in the observer error rate of below 4%. Similarly, if the HDL is aware that the distance between itself and its CPP is short, it can reduce the transmission power as to minimize its interference with other pairs in the neighborhood. To avoid overtaxing the microcontrolled device, the responsibility for implementing an intelligent power management scheme has been assigned to the CPP. The station can easily monitor the RSSI of the HDL as well as the error rate and instruct it to adjust the transmission power. A special CPP-to-HDL request has been provided for this purpose.

5 Conclusions

We have discussed the design of a low-cost wireless device for transmitting samples of medical data. One of the interesting aspects of our project was the rather idiosyncratic data transmission problem combining a flavor of streaming with the need for speedy and reliable delivery of all samples to the collection point. By understanding the limitations of the platform, we were able to accomplish our goal and make the best use of its small (but sufficient) resources.

From the commercial 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, as in the case considered in this paper, the RF bandwidth. Note that the latter 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 struggle and compete to deliver all the samples to their data-hungry 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.

References

[1] M. B. Abbott and L. L. Peterson. Increasing network throughput by integrating protocol layers. *IEEE/ACM Transactions on Networking*, 1(5):600–610, 1993.

[2] C. Barakat, E. Altman, and W. Dabbous. On TCP performance in a heterogenous network: a survey. *IEEE Telecommunications Magazine*, 38(1):40–46, 2000.

[3] D. Burton, H.O.; Sullivan. Errors and error control. *Proceedings of the IEEE*, 60(11):1293–1301, 1972.

[4] D. Chkliaev, J. Hooman, and E. de Vink. Verification and improvement of the sliding window protocol. In *Lecture Notes in Computer Science*, volume 2619, pages 113–127. Springer, 2003.

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

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

[7] L. Libman and A. Orda. Optimal sliding-window strategies in networks with long round-trip delays. *Computer Networks*, 46(2):219–235, 2004.

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

[9] H. Liu, H. Ma, and M. E. Z. S. Gupta. Error control schemes for networks: an overview. *Mobile Networks and Applications*, 2:167–182, 1997.

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

[11] A. Rahman and P. Gburzyński. Hidden problems with the hidden node problem. In *Proceedings of 23rd Biennial Symposium on Communications*, pages 270–273, Kingston, Ontario, Canada, May 29-June 1 2006.

[12] L. Song and D. Hatzinakos. A cross-layer architecture of wireless sensor networks for target tracking. *IEEE/ACM Transactions on Networking*, 15(1):145–158, 2007.

[13] O. Such et al. On-body sensors for personal healthcare. In B. Spekowius and T. Wendler, editors, *Advances in Health Care Technology: Shaping the Future of Medical Care*, volume 6, pages 436–488. Springer, 2006.

[14] Texas Instruments. CC1100 Single Chip Low Cost Low Power RF Transceiver, 2006. Revision 1.1.

[15] R. van Renesse. Masking the overhead of protocol layering. In *SIGCOMM '96: Conference proceedings on Applications, technologies, architectures, and protocols for computer communications*, pages 96–104, Palo Alto, CA, 1996.