On Reliable Wireless Streaming of Real-time Sensor Data

Agnieszka Boruta[1], Pawel Gburzynski[2] and Ewa Kuznicka[1]

[1]*Warsaw University of Live Sciences, ul. Nowoursynowska 166, 02-787 Warsaw, Poland*
[2]*Vistula University, ul. Stokłosy 3, 02-787 Warsaw, Poland*

Abstract
We discuss a practical problem related to wireless transmission of a continuous stream of readings from a sensor. The problem arose in the context of a contraption devised for monitoring the behavior patterns of working canines with the intention of spotting signs of their stress, exhaustion, or any indication of the animal's fatigue or discomfort that would call for the attention of its human companion. The device has been (is being) designed primarily as a vehicle for research in collaboration between the University of Live Sciences and Vistula University. The point we are trying to make is this paper is that tiny embedded systems aimed at specific applications within the realm of the so-called Internet of Things (IoT) come out best when built following a "holistic" approach. By "holistic" we mean taking into account, from the very bottom, the idiosyncratic aspect of the application instead of blindly relying on ready, layered, standardized, library solutions. In addition to reducing the footprint of the application, and enabling it to run in a cheaper and resource-frugal device, such an approach also translates into better performance. With the right selection of tools, it may in fact speed up the development process while resulting in a better quality of the product.

Keywords
wireless communication, remote sensing, wireless telemetry, streaming

1. The context

This paper deals with a subset of the technical aspects of a wider project aiming at researching simple and reliable automated methods for assessing the well-being of working dogs, including service dogs (military, police, disaster-response) as well as assistance dogs (guide, therapy). While the goals of our research probably need no long arguments in defense of their compassionate motivation, there are also solid remunerative reasons why an effective assessment of the animal's "quality" in providing its service matters. The dog training process is lengthy, complex, and expensive [1, 2] and good quality service/assistance dogs are extremely valuable [3, 4]. This stimulates studies along two lines: (1) to establish reliable criteria for early assessment of a dog's suitability for a particular kind of work/service [5, 6, 7, 8]; (2) to make sure that the animal is well taken care of and, in particular, any problems related to its work stress and generally health are quickly detected, diagnosed, and addressed [9, 10]. The latter issue can be reformulated as

29th International Workshop on Concurrency, Specification and Programming (CS&P'21)
agnieszka_boruta@sggw.edu.pl (A. Boruta); p.gburzynski@vistula.edu.pl (P. Gburzynski); ewa_kuznicka@sggw.edu.pl (E. Kuznicka)
0000-0002-3471-025X (A. Boruta); 0000-0002-1844-6110 (P. Gburzynski); 0000-0002-3760-1622 (E. Kuznicka)

that of an effective communication in the animal-to-human direction [11], as opposed to the more popular direction inherent in dog training.

While anomalies in a dog's behavior indicative of deficiencies in its well-being can be spotted by an expert human (veterinarian, behaviorist) through direct observation, our primary interest is in harnessing to this task sensing devices that, ideally, should be able to detect such problems automatically and signal them to the human supervisor. The issue can be viewed as falling under the more general domain of sensor-based diagnostics, e.g., similar to taking and interpreting ECG readings [12]. While it is obviously possible to subject the animal to extensive and authoritative veterinary assessments, including tests and sensor data collection interpreted by a human expert, we are interested in completely automated monitoring carried out by an inconspicuous and (basically) maintenance-free device being (basically) unnoticeable by the animal. Such a device, a wearable *Tag*, would be permanently carried by the dog, e.g., attached to a collar, and would convey wirelessly sensor data to a nearby access point for automated interpretation [13, 14].

The unobtrusiveness premise of the sensing/monitoring device trades off against the overt information content of the data that it can possibly collect. For example, it may seem worthwhile to try to obtain an EKG/ECG chart of the animal, which is a valuable source of information about the heart activity. One can expect such a chart to reasonably easily translate into a representation of the dog's tiredness or stress. While there exist experimental techniques for collecting this kind of data through "wearable" sensors [15], they overtax our premise by requiring direct access to the animal's skin. Even the less ambitious task of taking reliable heart rate without skin contact proves challenging [16].

Our long-term goal is to investigate how much one can accomplish with a completely unobtrusive device, requiring no skin contact and, preferably, no rigid attachment to the animal (in a specific position or place). The most natural sensor to try in this context is the Inertial Measurement Unit (IMU) being a combination of an accelerometer, a gyro, and a compass. Previous studies have reported various degrees of success in using the sensor (most notably the accelerometer component) for diagnosing various behavioral anomalies/problems in dogs [10, 17, 18]. We want to establish criteria for the classification of IMU data into simple signals indicative of some threshold levels of the animal's well-being in relation to its level of fatigue or stress that can be easily communicated to the human companion. The next step will be to built an actual practical device and application based on the outcome of our studies.

2. The setup

Our end goal being to fabricate a practically useful device, it makes sense to start with a view of the target application in mind. Ideally, we should use the same hardware for the experiments and for the final application. As the classification of the animal's activity patterns will be carried out based on the indications of an IMU, the embodiment of the sensor (the weight of the device and the mode of its attachment to the animal) is likely to matter because of its own (inherent) inertia component which will tend to influence the readings. This aspect of the project makes it similar to one of our earlier endeavors [19] where a series of research experiments carried out with an IMU-based sensor provided data to drive the design of a classification algorithm that could be subsequently implanted into the same device to a more practical end.

Having agreed that the experimental device should be identical to the target one, we still have to understand that the experimental version of the application (the software run by the device) is going to be drastically different from its target version. The role of the experiments is basically raw data collection. We want to amass a large amount of sensor readings, taken at the maximum rate that we can afford, from various representative animals acting under controllable conditions where experts can annotate the collected data with authoritative labels indicative of the dog's state. That data will be later used off-line to search for patterns that can guide the classification algorithms to be applied on the data collected by the target incarnation of the device. That part of the research methodology is beyond the scope of the present paper; we merely want to clarify the technical requirements for the experimental guise of the application.

The nature of the experiments, demanding that the animals act within environments appearing as close to their natural work conditions as possible, is an additional argument for the experimental devices being identical to the target ones, and it obviously precludes wired connectivity of the devices to external equipment. The footprint of the target device makes it impossible to store large volumes of data directly on it. Besides, the data must be annotated in real time, which makes it natural to use an external computer (a laptop or a tablet) for the actual collection and simultaneous annotation. Therefore, the sensor device is going to *stream* its readings wirelessly to the external computer. The streaming protocol is the aspect of the application discussed in the remainder of this paper.

The device used for data collection (and envisioned for the target application) is the CC1350 SensorTag[1] manufactured by Texas Instruments and featuring an ARM-based CC1350 microcontroller [20]. The SensorTag comes equipped with a number of sensors including the MPU9250 IMU by TDK InvenSense.[2] The complete device weights ca. 20 g (including a CR 2032 battery) and its dimensions are 44×32×6 mm. When attached to a dog's collar, it is not more obtrusive than a (slightly oversized) name tag.

The experimental setup consists of a pair of CC1350-based devices, one of them being the *Tag* worn by the dog, the other a CC1350 LaunchPad,[3] dubbed the *Peg*, connected over USB to the computer and acting as the RF access point (data sink) for the Tag. While the RF module of CC1350 can be configured to operate in Bluetooth mode, thus eliminating the need for a special access point, we opt for the so-called proprietary mode of the radio which gives us access to the raw channel. We prefer to circumvent the Bluetooth standard to: 1) increase the range of communication (to provide for a larger separation between the animal and the access point), 2) implement our private reliability scheme to increase the delivery fraction of the collected data. The proprietary mode operates within the 915 MHz ISM band offering parameterizable transmission power up to 14 dBm and (raw) transmission rates up to 500 kbps. Data exchanged over the RF channel is organized into packets with the maximum packet length (practically) limited to 60 bytes. The Bluetooth capability of the device will become handy in the target application where the device will be able to communicate directly with a smartphone. The essential classification will then be carried out in the Tag [19], so there will be no need for reliable streaming of large volumes of readings to the access point.

[1]https://dev.ti.com/cc1350stk

[2]https://invensense.tdk.com/products/motion-tracking/9-axis/mpu-9250/

[3]http://dev.ti.com/launchxl-CC1350

3. The problem

In its most general formulation, independent of the hardware and application context, the problem is that of implementing reliable communication across an unreliable channel. We deal with an asymmetric link where the Tag continuously sends a stream of data to the Peg. Ideally, we would like *all* the data generated by the Tag to (eventually) reach the Peg, in the proper order, such that the complete, timed, and annotated stream of sensor readings produced by the Tag is stored at the collection computer.

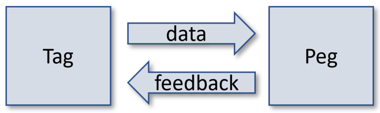


Figure 1: The channel model

Standard solutions to this problem involve a feedback channel operating in the Peg-to-Tag direction, as shown in Figure 1. The simplest of them has been known as the Alternating Bit Protocol [21, 22] and consists in explicitly acknowledging every single packet received by the Peg from the Tag. Various generalizations and improvements upon this simple scheme, including the one underlying TCP, are known under the name of ARQ protocols [23, 24]. Their primary objectives are: 1) to reduce the amount of feedback traffic, 2) to improve the continuity of the forward traffic when the losses are low and/or the bandwidth-delay product of the link is large [25].

The problem of *absolutely* reliable data transmission primarily concerns information whose utmost integrity is essential, i.e., the transmission of files. With streaming, the data is assumed to be created continuously and the problem of its reliable delivery receives a different flavor. In many cases the information is not stored at the source in the form of a complete file whose missing fragments could be requested at random by the recipient and retransmitted later. But even if this happens to be the case, the issue of its reliable delivery is conditioned by the real-time character of the reception where the data often become obsolete and useless if not delivered within a certain time window (like in streaming of voice and/or video). Consequently, while standard streaming applications may insist on high-quality of delivery, they typically are (and usually must be) prepared to deal with acceptable (innate) data loss [26, 27].

Our problem is specific in that: 1) the information collected by the Tag cannot be entirely stored at the source, so the availability of its undelivered fragments is restricted; 2) there is no issue of real-time playback at the recipient (unlike in streaming audio or video); 3) we can be prepared to deal with (acceptable) losses (and seemingly have to because of 1). The problem is reminiscent of one we dealt with before [28] (the hardware context was similar), and it may be worthwhile to emphasize the difference. In the case of [28] the entire collected data set was stored at the source Tag, and the problem was formulated as minimizing the time of its *complete* delivery to the Peg (the completeness of delivery was essential, so we were basically interested in reliable and fast file transfers). In the present case, we are inclined to deal with occasional data loss (the stream is infinite for all practical purposes) and the issue is to maximize the delivery fraction thus maximizing the feasible data collection rate.

If losses are unavoidable, one can often simplify the solution by eliminating the feedback channel altogether focusing instead on enhancing the reliability of communication in the physical layer, e.g., through forward error correction (FEC) techniques [29]. The proprietary mode of the RF module of CC1350 comes with a number of options that can be applied to this end. They effectively trade the transmission range and bit rate for reliability and amount to an important set of parameters that have to be tuned for best performance regardless of any other tools. However, depending solely on them would be too restrictive from our point of view. The dynamic nature of the propagation environment, including the variable distance between the Tag and the Peg, would lock the channel into a conservative setting, offering an acceptable (or passable) loss rate for the worst case scenario, while unnecessarily reducing the opportunities for collecting more data in a friendlier environment. Our hope is to achieve a much better flexibility with a properly designed feedback scheme: the bandwidth to be sacrificed for reliability will only be sacrificed when needed.

In a high-level discussion of ARQ schemes (and in many implementations of such schemes in the wired world) it is assumed that the feedback channel (Figure 1) is separate from the forward (data) channel. This is to say that the feedback messages do not disrupt the data stream and, in particular, they can be sent at any rate up to some maximum with their impact being solely positive. This is seldom true in wireless communication. Even if the two channels are in fact separate, which they mostly aren't, they will tend to interfere. Generally, setting aside a sizable portion of the RF bandwidth for a "frivolous" feedback channel makes that bandwidth unavailable for the *proper* use which is transmitting data. For example, in Bluetooth, e.g., within the framework of an ACL link [30], the essentially single channel must be partitioned (time-divided) into two parts to provide for two-way communication. Realizing that the feedback channel is going to directly coexist with the forward data channel, we would like to make it flexible and, in particular, avoid a rigid, time-division-based pre-allocation of bandwidth for the two channels. Our goal is to try to reduce the impact of the feedback channel on data bandwidth to the minimum needed by the application and demanded dynamically by the temperamental RF medium.

4. The solution

We propose a protocol for improving the reliability of conveying sensor data from the Tag to the Peg. The improvement will be evaluated in reference to the reliability achievable with purely hardware means, by assuming a unidirectional data channel (no feedback). The solution illustrates how application constraints can influence the design of low-level communication schemes stimulating a holistic approach to programming the application.

Owing to the lack of real-time requirements, the possibility of unrecoverable losses results solely from the finite buffers at the Tag. Whatever buffer space is available will be allocated to a shifting window of the collected data. The Peg will be able to request retransmission of those lost packets that are still present within the window.

The **Tag side** of the protocol is described by two threads: the generator of blocks of sensor readings (dubbed the generator thread) and the transmitter of those blocks on the RF channel. The blocks are stored in a singly-linked queue, denoted by Q and depicted in Figure 2, whose

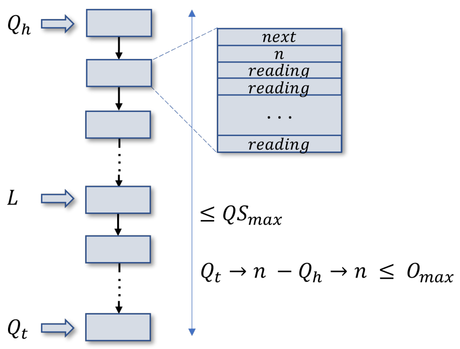

Q_h
L
Q_t
next
n
reading
reading
...
reading
$\leq QS_{max}$
$Q_t \rightarrow n \; - Q_h \rightarrow n \; \leq \; O_{max}$

Figure 2: The block queue

size is limited. The queue is represented by two pointers: Q_h (the head), and Q_t (the tail). When Q is empty, we have $Q_h = Q_t = null$. One block contains readings to be expedited in a single RF packet. In addition to the readings (whose number is the same for all blocks), a block b contains a link to the next block in Q (or *null* if the block is the tail one) and the sequence number of the block in the stream, which we shall denote by $b\rightarrow n$. This number starts with 1 (for the first block generated in a session) and is incremented by 1 for every new block issued by the thread, as explained below. The maximum size of the queue (i.e., the maximum number of blocks that it can contain at a time) is determined by the amount of storage available at the Tag and denoted by QS_{max}. QS_{max} can be assumed to be (roughly) equal to M/s) where M is total amount of RAM available at the Tag for storing Q and s is the (fixed) block size. When the streaming operation starts, Q is initialized to empty and N (the current block number) is initialized to 1.

4.1. The generator thread

Every $1/f_s$ seconds, where f_s is the sampling frequency, a sensor reading is taken. The reading is stored in the *current buffer* denoted by *CB*. *CB* is filled by consecutive readings (f_s times per second) until it becomes complete (its capacity is reached).

When *CB* becomes complete, the thread executes a function named *add_CB* which sets $CB\rightarrow n$ to N, increments N by 1, and appends *CB* at the end of Q (updating Q_t and possibly Q_h as needed). Before adding the new block to the queue the function makes sure that, after the addition of *CB*, Q will not exceed its two limitations (of which both are necessary). **Limitation 1** says that the total number of blocks in the queue is never bigger than QS_{max}. **Limitation 2** requires that $Q_t\rightarrow n \; - \; Q_h\rightarrow n$, i.e., the difference between the first and the last block number in the queue, be less than or equal to O_{max} which we call the *maximum block offset*. The first limitation simply makes sure that Q never exceeds its allotted storage. The second limitation constrains the age difference of the blocks kept in the queue. This is needed for two reasons. The first (informal) reason is that, in the face of the storage limitations, it makes little sense to keep around old blocks that (for one reason or another) have not made it to the collection point. The second reason is that we want to be able to reference past blocks relative to the current place (block number) within the session, for which we want to restrict the range of requisite offsets.

Note the *CB* is appended at the tail of Q becoming new Q_t. To enforce the limitations, *add_CB* examines the block at the front of Q (pointed to by Q_h) and discards it for as long as any of the two limitations is violated when *CB* is included the queue. One invariant of the queue is that the numbers of blocks stored in it are strictly increasing (they need not be consecutive as we shall shortly see). Thus, looking at Q_h and the total number of blocks stored in Q is enough to verify the limitations. Having added *CB* to Q, the function opens a new empty version of *CB* which the thread will now be filling from scratch.

The blocks stored in Q will be transmitted to the Peg by the second thread (as explained below), but not immediately discarded, unless new blocks, containing most recent readings, cannot be accommodated into Q because of the limitations. When that happens, the sampling thread will be removing blocks from the front of the queue thus giving preference to fresh readings.

4.2. The transmitter thread

The thread operates in rounds where it transmits a *train* of blocks to the Peg and then *reverses the channel* for a short while [28] to allow the Peg to acknowledge the train. Once the transmission of a train is started, it is carried out back-to-back with a minimum inter-packet spacing, just to enable the recipient to accept the individual packets from the channel.

A train always consists of the same number of packets (blocks) which we shall denote by T. When commencing a train, the thread starts by scanning consecutive packets from Q (beginning from Q_h) and transmitting them in sequence. When the last packet (the one pointed to by Q_t) has been handled and the train is still incomplete, the transmitter thread will simply wait until the generator thread delivers the next block, i.e., the remaining packets in the train will be sent as new blocks materialize in Q. Note that the blocks are *not* removed from Q as they are being transmitted.

A train packet contains the block number $b{\to}n$ of the block carried by the packet, and the packaged sensor readings copied from the block. The block number always allows the Peg to authoritatively place the contents of a received packet within the complete stream of readings, regardless of how many packets have been lost and how the received packets have been misordered with respect to the original stream.

Having completed the current train, the transmitter thread enters a loop in which it expects to receive a response (an acknowledgment packet) from the Peg. Within that loop, the thread periodically sends a short EOT (end of train) packet (to make sure that the Peg has recognized that the train has ended and a response is expected) and waits for a short while for the ACK (which should normally arrive after a minimum delay). The EOT packet carries three items of information: the train number modulo 256 (a single byte) used to match trains to acknowledgments, the number of the last block transmitted in the train (denoted by L), and the back offset (O_b) from L to the oldest packet still held in Q ($O_b = L - Q_h{\to}n + 1$).

The idea is that having received an EOT packet, the Peg can assess which blocks have been missing (it knows the number of the last block sent by the Tag) and it also knows the minimum number of the block that it can still ask the Tag to retransmit. The reason why the latter is specified as an offset with respect to L is technical: the offset information can be conveyed in two bytes instead of four (needed for a full block number). It illustrates one practical and seemingly mundane aspect of real life in the embedded world where it always makes sense to try to save on individual bytes. Note that the generator thread enforces a limitation reducing the maximum difference between the numbers of blocks stored in Q.

The number of the oldest block still available in Q conveyed by the Tag in the EOT packet reflects the state of Q at the moment the EOT packet is constructed and scheduled for transmission. This information may become outdated by the time the Peg builds and transmits its response and, more importantly, by the time that response arrives and is interpreted by the Tag, because Q can be trimmed by the generator thread independently of the transmitter thread. This is OK. The possibility that a block may be irretrievably lost is factored into the scheme. It can happen that the Peg asks for the retransmission of a block that is no longer available. At the end of the next train the Peg will learn (from the new EOT packet) that the block is no more, so it will know that it makes no sense to keep asking for it.

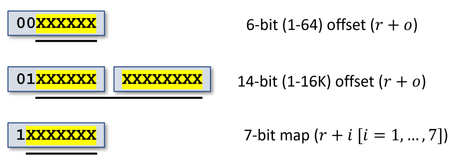
The transmitter thread will keep retransmitting the EOT packet (possibly updating its parameters) until an acknowledgment arrives from the Peg. Normally this should happen right away, but in a pathological scenario, if the connectivity has been broken for a long time, the contents of Q may evolve to the point where L is no longer available. This is why the minimum value of O_b for the situation when Q still contains the block number L is 1. When O_b in an EOT packet is 0, it means that none of the blocks up to (and including) the end of the last train is available any more, so the Peg need not ask for any retransmissions.

4.3. The acknowledgment

The role of the acknowledgment (ACK) packet is to indicate to the Tag which blocks have been missing with respect to the end of the last train and relative to the Peg's knowledge regarding the blocks that it can still hope to receive. Again, mundane technical constraints force us to be frugal about the representation of information within the ACK. For one thing, the entire message should fit into a single packet whose useful (payload) size is limited to 60 bytes. Organizing the ACK message into multiple packets would introduce obscure reassembly problems complicating things beyond practical [28]. Consequently, the ACK format should allow the Peg to maximize the population of (independent) block numbers that it can specify in a single packet to cover worst-case scenarios and, more generally, to minimize the size of the (typical) ACK packet. As the ACK traffic *interferes* with an otherwise smooth series of back-to-back transmissions of useful data (causing channel reversals [28]), its impact (and the incurred reduction of bandwidth) should be minimized.

One mandatory item carried in an ACK packet is the train number (modulo 256) to which the acknowledgment applies. Any other data included in the packet pertain to the blocks that the transmitter thread of the Tag should *retain* in Q before commencing the next train. In other words, these are the blocks that the Peg wants retransmitted. If the ACK contains no data beyond the single mandatory byte (which is the ideal case), the message to the Tag is simple: erase Q up to and including L, i.e., the end of the last train.

The structure of an ACK packet is best explained by the way the information is interpreted by the Tag upon its arrival. The interpretation is carried out by function *handle_ACK* invoked by the transmitter thread. Following the train number stored as the first byte of the packet, the function interprets consecutive bytes as descriptors of the blocks that have been missing by the Peg and, if possible, should be retransmitted. Those blocks are specified within the ACK packet



00XXXXXX
6-bit (1-64) offset ($r + o$)
01XXXXXX
XXXXXXXX
14-bit (1-16K) offset ($r + o$)
1XXXXXXX
7-bit map ($r + i$ [$i = 1, \dots, 7$])

Figure 3: Descriptors of missing blocks

in the increasing order of their numbers.

While interpreting the descriptors, the function stores in r the last block number mentioned by a previous descriptor, to be used as a reference. The value of r is initialized to $L - O_{max} - 1$ where O_{max} is the maximum legal difference between block numbers in Q (see above). The ACK bytes are interpreted in the following way (see Figure 3):

1. If the two most significant bits of the byte are zero, then its remaining six bits are interpreted as a forward offset from r minus 1, i.e., r is incremented by the nonnegative integer value stored on those bits plus 1, and block number r is marked to be retained in Q. Note that the minimum sensible value of an offset is 1.
2. If the two most significant bits of the byte are 01, then the remaining six bits of the byte are prepended (on the left) to the next byte and the two bytes form together a 14-bit (forward) offset from r minus 1.
3. If the most significant bit of the byte is 1, then the remaining seven bits are treated as a bit map, each bit indicating an individual block relative to the value of r. For this purpose, the bits are numbered from 1 to 7 (right to left), and when bit number i is set, the block number $r + i$ is marked to be retained. At the end of processing the byte, r is set to $r + 7$, i.e., the last block number covered by the bit map, regardless of whether the block was marked as retained or not.

The important point is that the interpretation of the consecutive bytes, starting from the front of the packet's payload, produces increasing values of r which eases the operation of updating Q. The queue is scanned in place, in the most natural manner, starting from Q_h, and any blocks whose numbers are not mentioned in the ACK are discarded. Before interpreting the first byte of the ACK, *handle_ACK* initializes r to $L - O_{max} - 1$ to provide a sensible, default, initial value (so the first byte of the ACK can be a forward offset). It might seem natural to initialize r to $L - O_b$ (based on the value of O_b passed in the EOT packet), which would make the range of the initial offset better contained. However, while the value of L is nailed to the train, O_b may change in the different (retransmitted) versions of the EOT packet for the same train. As the Tag cannot know which particular copy of EOT served the Peg as the basis for its ACK, O_b is not a well-known value that both parties can always agree on.

Following the reception of an ACK packet from the Peg, the transmitter thread will start the next train with a trimmed-down version of Q. The queue will have been emptied of all blocks that 1) have numbers less than or equal to L from the previous train, and 2) have *not* been mentioned in the ACK packet.[4]

[4]Strictly speaking, the packet carries a *negative* acknowledgement.

Table 1
A tentative setting of protocol parameters

Parameter	Value	Units
Channel transmission rate	50	kbps
Samples per block	12	3-vectors
Max. Q size: QS_{max}	128	blocks
Train length: T	64	packets
Max. block offset: O_{max}	2047	blocks
Packet space (within train)	5	ms
End of train space (for the ACK)	20	ms

4.4. The Peg

The device passes the received blocks to the computer, locally keeping track of the holes in the received sequence, down to the maximum negative offset from the end of the last train. The blocks are tallied in a bit map whose fixed size covers the interval O_{max}. For the ease of calculations, the actual momentary coverage of the map is described by the current *base* block number B which is shifted to the newest value of $L - O_b$ learned by the Peg. As B is updated, the (logical) beginning of the bit map shifts automatically (in a circular fashion), so the bit map itself is not shifted (no copying is involved), except for clearing the obsolete entries at the tail.

Having received an EOT packet, the Peg updates the base of its bit map to $L - O_b$, sets its reference block number r to $L - O_{max} - 1$, and begins constructing the consecutive bytes of the ACK packet. Given the next missing block to be accounted for, there are these possibilities which are greedily examined in this order: 1) the last entry in the ACK packet is a bit map byte and the block number falls under its coverage, 2) the block number is within a bit-map range from the current value of r, 3) the block number can be represented by a short offset from r, 4) a long (two byte) offset is needed. In the first case, the block is simply added to the bit map byte without extending the ACK packet. In the second case, if the difference between the block number and r is less than 7, a new bit map byte is added to the packet. Note that a bit map byte comes at the same storage expense as a short offset from r, so there is no point in looking ahead (a greedy approach works fine). Preference is given to the bit map when, based on the current block number alone, the map stands a chance of accommodating at least one more block.

In the unlikely case when the ACK packet becomes filled up to the limit of its size, the receiving Tag will assume that all the blocks falling behind the last block number represented in the ACK are implicitly marked as missing. Note that this will only happen in highly abnormal conditions where pessimistic assumptions regarding the unknown are probably warranted.

5. Performance

The first implementation of our scheme addresses a planned series of experiments with animals aimed at collecting 128 acceleration samples per second. Our intention was to tune the parameters of the protocol until we get a satisfying performance for the task at hand. Table 1 lists the numerical values of the parameters assumed for the initial tests.

While the values in Table 1 must be treated as tentative, they were produced by confronting our expectations with the parameters and capabilities of hardware. One sample of acceleration amounts to three scalars which we pack into 30 bits (10 bits per value). With some modest creativity, a 50-byte packet encodes 12 samples plus the 32-bit block number. The collection rate of 128 samples per second translates into about 11 packets per second, the total (transmitted) length of every packet being 60 bytes. This implies the (continuous) rate of 5280 bits per second and clearly suggests that the raw RF channel bandwidth of 50 kbps is more than sufficient to accommodate the transfers.

Our experiments have demonstrated, at first sight somewhat surprisingly, that it is virtually impossible to lose data in a streaming session for as long as the session operates within the framework of raw technical feasibility. We can easily cater to sampling frequencies up to 512 samples per second (which is just one notch below the maximum capacity of the the sensor) without increasing the channel rate, and up to the maximum of 1024 samples per second at a slight increase of the channel rate, practically without losing *any* samples!

This can be argued quite formally. Let R denote the raw rate at which blocks can be transmitted, back-to-back, assuming smooth operation and no errors (we shall ignore the ACKs for a while). Let r_a denote the target effective rate corresponding to the frequency at which we would like to reliably collect samples of sensor readings. Let r_e be the block error rate of the channel. The system can be modeled as a server shown in Figure 4.

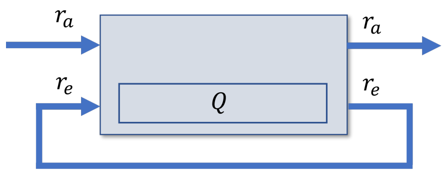


Figure 4: The performance model

The bottom path represents the traffic incurred by errors and the consequent retransmissions requested by the Peg in its acknowledgments. This is what the queue Q is really for: to accommodate blocks that have to be retransmitted because of errors.

Consider the system at equilibrium and note that the upper path is stable and deterministic: blocks arrive at a steady rate r_a, their processing time is fixed, and they leave the server at the same rate. Consequently, we can ignore the dynamics of the upper part assuming that its impact consists in removing from R a fixed portion amounting to r_a. Whatever is left, i.e., $R - r_a$ can be treated as the bandwidth available for the bottom part of the traffic, i.e., for retransmissions.

Suppose that the errors are independent and they occur at the same probability P_e for every transmitted block. Then we have:

$$r_e = r_a \times \sum_{i=1}^{\infty} P_e^{\,i} = \frac{r_a \times P_e}{1 - P_e} \tag{1}$$

The bottom part of the service can be approximated as an M/D/1 queue where $\rho = r_e/(R - r_a)$ (the utilization parameter) indicates the fraction of the spare bandwidth (whatever remains after accounting for r_a) taken by the retransmissions. The expected occupancy of the queue is then:

$$C = \rho + \frac{1}{2}\left(\frac{\rho^2}{1-\rho}\right) \tag{2}$$

The graph of C versus ρ (Figure 5) is quite illuminating. It shows that unless the utilization factor becomes very close to unity, i.e., the retransmissions fill all the bandwidth left to them, the demand for queue space is extremely modest. Consequently, to see the queue overflow (for any size above the train length T), and actual losses start to materialize, we have to bring the system to the very edge of its equilibrium. Then, of course, there is no surprise that the system refuses to cooperate: it could not possibly do any better under the best possible scheme.

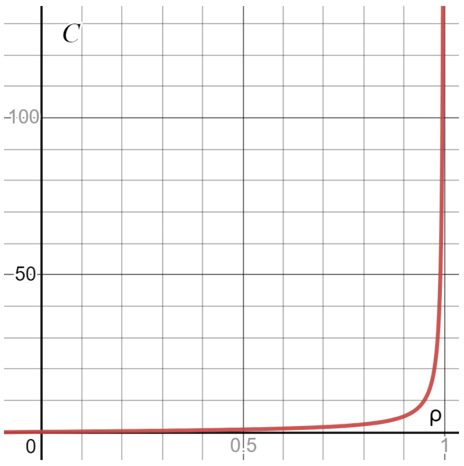


Figure 5: C = the average occupancy of Q versus the utilization factor ρ

The above model is oversimplified a bit by its abstraction from the acknowledgments. Their main impact is in stealing a fraction of R proportional to the total portion of the bandwidth used by the trains. To factor them in, we should express the utilization parameter as:

$$\rho = \frac{r_e}{R - r_a - f_A \times (r_e + r_a)} \tag{3}$$

where f_A is the amount of bandwidth (expressed in blocks) used up by the exchange of one acknowledgment.

6. Summary

We have presented a protocol for reliable streaming of telemetric data over an unreliable wireless channel. Our scheme seems to make a good use of the available bandwidth, especially in the specific context of its inspiring application. On the sender's side, this is accomplished by an efficient organization of storage for the outstanding (unacknowledged) packets. The feedback sent by the recipient is minimized to reduce the impact of channel reversals on the bandwidth available for the forward traffic. The proposed scheme is intended for small-footprint wireless sensing devices where the amount of memory for packet buffers is drastically limited.

References

[1] B. J. Cooke, L. B. Hill, D. P. Farrington, W. D. Bales, A beastly bargain: A cost-benefit analysis of prison-based dog-training programs in Florida, The Prison Journal 101 (2021) 239–261.

[2] R. A. Yount, M. D. Olmert, M. R. Lee, Service dog training program for treatment of posttraumatic stress in service members., US Army Medical Department Journal (2012).

[3] G. Lippi, G. Cervellin, M. Dondi, G. Targher, Hypoglycemia alert dogs: a novel, costeffective approach for diabetes monitoring?, Alternative therapies in health and medicine 22 (2016) 14.

[4] R. Schoenfeld-Tacher, P. Hellyer, L. Cheung, L. Kogan, Public perceptions of service dogs, emotional support dogs, and therapy dogs, International journal of environmental research and public health 14 (2017) 642.

[5] G. S. Berns, A. M. Brooks, M. Spivak, K. Levy, Functional MRI in awake dogs predicts suitability for assistance work, Scientific reports 7 (2017) 43704.

[6] E. E. Bray, K. M. Levy, B. S. Kennedy, D. L. Duffy, J. A. Serpell, E. L. MacLean, Predictive models of assistance dog training outcomes using the canine behavioral assessment and research questionnaire and a standardized temperament evaluation, Frontiers in veterinary science 6 (2019) 49.

[7] C. Byrne, J. Zuerndorfer, L. Freil, X. Han, A. Sirolly, S. Cilliland, T. Starner, M. Jackson, Predicting the suitability of service animals using instrumented dog toys, Proceedings of the ACM on Interactive, Mobile, Wearable and Ubiquitous Technologies 1 (2018) 1–20.

[8] J. M. Slabbert, J. S. Odendaal, Early prediction of adult police dog efficiency—a longitudinal study, Applied Animal Behaviour Science 64 (1999) 269–288.

[9] M. Brložnik, V. Avbelj, A case report of long-term wireless electrocardiographic monitoring in a dog with dilated cardiomyopathy, in: 2017 40th International Convention on Information and Communication Technology, Electronics and Microelectronics (MIPRO), IEEE, 2017, pp. 303–307.

[10] G. J. Jenkins, C. H. Hakim, N. N. Yang, G. Yao, D. Duan, Automatic characterization of stride parameters in canines with a single wearable inertial sensor, PloS one 13 (2018) e0198893.

[11] J. M. Alcaidinho, The internet of living things: enabling increased information flow in dog-human interactions, Ph.D. thesis, Georgia Institute of Technology, 2017.

[12] M. Brložnik, Š. Likar, A. Krvavica, V. Avbelj, A. Domanjko Petrič, Wireless body sensor for electrocardiographic monitoring in dogs and cats, Journal of Small Animal Practice 60 (2019) 223–230.

[13] Pet Pace Pets Remote Monitoring System, User Manual, PetPace Ltd., 2020. URL: https://petpace.com/.

[14] Animo Quick Start Guide, SureFlap Ltd., 2019. URL: https://www.surepetcare.com/en-au/animo.

[15] M. Foster, R. Brugarolas, K. Walker, S. Mealin, Z. Cleghern, S. Yuschak, J. Condit, D. Adin, J. Russenberger, M. Gruen, et al., Preliminary evaluation of a wearable sensor system for heart rate assessment in guide dog puppies, IEEE Sensors Journal (2020).

[16] M. Foster, S. Mealin, M. Gruen, D. L. Roberts, A. Bozkurt, Preliminary evaluation of a

wearable sensor system for assessment of heart rate, heart rate variability, and activity level in working dogs, in: 2019 IEEE SENSORS, IEEE, 2019, pp. 1–4.

[17] F. M. Duerr, A. Pauls, C. Kawcak, K. K. Haussler, G. Bertocci, V. Moorman, M. King, Evaluation of inertial measurement units as a novel method for kinematic gait evaluation in dogs, Veterinary and Comparative Orthopaedics and Traumatology 29 (2016) 475–483.

[18] M. Foster, J. Wang, E. Williams, D. L. Roberts, A. Bozkurt, Inertial measurement based heart and respiration rate estimation of dogs during sleep for welfare monitoring, in: Proceedings of the Seventh International Conference on Animal-Computer Interaction, 2020, pp. 1–6.

[19] E. Kuźnicka, P. Gburzyński, Automatic detection of suckling events in lamb through accelerometer data classification, Computers and Electronics in Agriculture 138 (2017) 137–147.

[20] Texas Instruments, CC1350 SimpleLink Ultra-Low-Power Dual-Band Wireless MCU, 2019. URL: http://www.ti.com/lit/ds/symlink/cc1350.pdf, technical document SWRS183B.

[21] K. A. Bartlett, R. A. Scantlebury, P. T. Wilkinson, A note on reliable full-duplex transmission over half-duplex links, Communications of the ACM 12 (1969) 260–261.

[22] W. Lynch, Reliable full-duplex transmission over half-duplex telephone lines, Communications of the ACM 11 (1968) 407–410.

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

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

[25] T. Lakshman, U. Madhow, The performance of TCP/IP for networks with high bandwidth-delay products and random loss, IEEE/ACM transactions on networking 5 (1997) 336–350.

[26] C. Baransel, W. Dobosiewicz, P. Gburzyński, Routing in multi-hop switching networks: Gbps challenge, IEEE Network Magazine (1995) 38–61.

[27] R. Pereira, E. G. Pereira, Video streaming considerations for internet of things, in: 2014 International Conference on Future Internet of Things and Cloud, IEEE, 2014, pp. 48–52.

[28] P. Gburzynski, B. Kaminska, A. Rahman, On reliable transmission of data over simple wireless channels, Journal of Computer Systems, Networks, and Communications 2009 (2009).

[29] A. Nafaa, T. Taleb, L. Murphy, Forward error correction strategies for media streaming over wireless networks, IEEE Communications Magazine 46 (2008) 72–79.

[30] Bluetooth SIG, Bluetooth technology, 2020.