Torfs and Blondia *EURASIP Journal on Wireless Communications and Networking* 2011, **2011**:37
http://jwcn.eurasipjournals.com/content/2011/1/37

EURASIP Journal on Wireless Communications and Networking
a SpringerOpen Journal

RESEARCH Open Access

Analysis of TDMA scheduling by means of Egyptian Fractions for real-time WSNs

Wim Torfs* and Chris Blondia

Abstract

In Wireless Sensor Networks (WSNs), Time Division Multiple Access (TDMA) is a well-studied subject. TDMA has, however, the reputation of being a rigid access method and many TDMA protocols have issues regarding the entering or leaving of sensors or have a predetermined upper limit on the number of nodes in the network. In this article, we present a flexible TDMA access method for passive sensors, that is, sensors that are constant bitrate sources. The presented protocol poses no bounds on the number of nodes, yet provides a stable framing that ensures proper operation, while it fosters that every sensor gets its data on time at the sink and this in a fair fashion. Even more, the latency of the transmission is deterministic and thereby enabling real-time communication. The protocol is developed, keeping in mind the practical limitations of actual hardware, limiting the memory usage and the communication overhead. The schedule that determines when a sensor can send can be encoded in a very small footprint and needs to be sent only once. As soon as the sensor has received its schedule, it can calculate for the rest of its lifetime when it is allowed to send.

I. Introduction

A Wireless Sensor Network (WSN) is an interesting type of network which can be used for several objectives. For instance, data monitoring is such application, where sensors send data at regular intervals. Such networks consist of devices that are considered to be small, low cost and with limited resources, such as a low amount of working and program memory, low processing power and a low battery capacity. Such kind of networks are presumed to work in an unattended fashion and it is often difficult or labor intensive to provide any maintenance to the sensors.

It is a challenge to perform monitoring as efficiently as possible due to the limited resources available in such sensors. Since the sensors need to work in an unattended fashion, it is favored that the battery lifetime is as large as possible. However, data should be sent at regular intervals, with the exception of event monitoring where data is transmitted only if an event has been positively identified. Moreover, lengthy processor intensive calculations, such as complex data processing, are discouraged due to the drainage of the battery. Therefore, we focus our research on the continuous monitoring applications where no preprocessing of the sampled data is performed on the sensors. As a consequence, every sensor can be considered as a constant bitrate source, of which the bitrate depends on the type of sampled data. This results in a heterogeneous WSN that needs to be able to cope with different rates in a flexible manner.

Algorithms specifically designed for WSNs, should enable a sensor to enter a sleep state on a regular basis to limit the battery drainage and hence preventing idle listening and overhearing. Collisions during the transmission of packets should be prevented, since a retransmission leads to waste of battery power. TDMA is a class of protocols that not only avoids collisions, but also provides a sleep schedule. However, there are a few issues concerning the use of TDMA in WSNs.

First, a WSN needs to be flexible with regard to the number of sensors and the heterogeneous properties of the network. TDMA on the other hand makes use of a rigid frame structure. A variable slot size or a variable number of slots in a frame is not desirable because of this strict schedule that needs to be followed by every sensor. Changing the slot size or number of slots every frame, amounts to passing a new schedule to all nodes every frame. Keeping in mind that the wireless medium is lossy, there is no guarantee that all sensors adopt

* Correspondence: wim.torfs@ua.ac.be
University of Antwerp-IBBT, Middelheimlaan 1, 2020 Antwerp, Belgium

the same schedule since they might have missed its announcement.

Secondly, TDMA-based protocols often pose an upper limit on the number of sensor that can be supported in the network. A protocol for a WSN should not have such bounds. The area of interest where monitoring is provided should be easy to extend, without any limitation on the maximum number of sensors.

We propose a TDMA scheduling algorithm that complies to the characteristics of both, that is, it is flexible, but also makes use of a rigid framework. By means of Egyptian Fractions and binary trees, we can compose a TDMA schedule that allows sensors to send in specified slots during certain frames, just enough to guarantee their required bandwidth and hence minimizing the battery drainage. This schedule is periodic, resulting in a TDMA schedule that needs to be sent only once, which leads to a low protocol overhead. The protocol poses no boundary on the number of nodes, only the available bandwidth provides an upper bound. Due to the specific construction of the schedule, additional bandwidth allocations do not require other sensors to adjust their schedule. A supplementary property of the schedule is that the latency is perfectly predictable, which means that the protocol is suited for real-time applications.

One of the goals is to keep the protocol as realistic as possible, taking into account the hardware limitations such as limited memory and processing power. To prove the previous statement, an actual implementation of the protocol on Tmotes was described in our previous paper [1]. It describes superficially the protocol itself, and is more focused on an actual implementation of the protocol than the analysis of the operation of the protocol. On the contrary, in this article, we provide an extensive explanation regarding the internal operation of the protocol. Furthermore, this article analyzes in detail the theoretical real-time behavior of our protocol and deduces a formula that predicts the latency that can be expected. The measurements in [1] verify whether this formula also applies when using a practical implementation.

In the next section, some of the related work is described. The third section presents the algorithm. After that, a thorough analysis of the algorithm is given, based upon a perfect node and traffic. The fifth section describes the effects of a bursty arrival of the data. And the last section concludes our findings.

II. Related work

Energy efficiency is a frequently discussed topic in protocols for WSNs, such as S-MAC [2] and T-MAC [3], where the available time is split up in an active time and a sleep time. During the active time, the protocols use the standard CSMA method to communicate. As a result, these protocols still have problems regarding overhearing, idle listening and collisions during their active periods. TDMA-based protocols, such as L-MAC [4], A-MAC [5], a dynamic TDMA scheme [6] and pedamacs[7], do not have such issues. These protocols use the wireless medium only when it is required to receive or send data. Otherwise, their transceivers do not need to be enabled. The problem is that these protocols are designed for a certain purpose. In other situations, these protocols might not behave as well as they were designed for. The biggest issue posed by these schemes is while considering energy efficiency, the actual required throughput is neglected, where a large amount of energy can be saved.

Our algorithm allows every sensor to use the wireless medium for a time, proportional to its requested bandwidth. The Weighted Fair Queuing (WFQ) [8-10], also known as packet-by-packet generalized processor sharing (PGPS) [11], provides the capability to share a common resource, and gives guarantees regarding the bandwidth usage. WFQ is a widely referred to protocol in the scheduling theory to achieve a fair schedule. Our algorithm uses a fractional representation of the requested bandwidth in order to determine the number of resources. WFQ uses a comparable method, as it is a packet approximation of Generalized Processor Sharing (GPS) [12], where every session obtains access to the resource, but only for $1/N$th of the bandwidth, where $\boldsymbol{N}$ represents the available bandwidth divided by the requested bandwidth.

In [13], the requested bandwidth is also split up according to some common factor, which forms the key to find a schedule. The schedule is used to create an allocation pattern, such that the obtained rate of the allocation is larger than the requested rate. The scheduling itself is done by means of Earliest Deadline First (EDF) scheduling.

[14] claims too that bandwidth is being wasted by too large slots. The concept of a shared real-time communication channel is introduced in this article. The slots that belong to such a shared channel can be used by a certain number of senders. These senders have the right to send data during this slot. In order to resolve conflicts between the senders, the authors rely on the underlying multiple access bus.

The concept of scheduling resources fractionally is also used in [15], a protocol designed for video conferencing.

In [16], we have already presented the basic idea for the protocol described here, that is, the time divisioned usage of a slot by multiple sensors. By means of calculation of the common factor between requested bitrates, a scheduling scheme can be found that allows the sharing of a single slot by multiple nodes through a round-robin like scheme. We proposed in this article to use the greatest common divider (gcd) as a common factor, which is a valid solution for bitrates that have a low least common multiple (lcm). However, since the periodicity is determined by the value of the lcm, it results in far too big cycles when the gcd is significantly smaller compared to the bitrates. Another disadvantage of this method is shown in [17], where it is mentioned that round-robin scheduling results in a fair schedule if all data amounts are equally sized. If they vary too much, nodes with more data are favored over others.

GMAC [18] is a protocol that utilizes the geographical position of its two-hop neighbors. It makes use of a technique comparable to our algorithm to share the medium by allowing nodes to use a certain slot in specified frames. It defines a superframe, which is split up into *c* cycles. Each of these cycles is then split up into *s* slots. One cycle represents a rotation in a geometric circle, that is, every slot represents $\frac{360}{s}$ degrees. A requirement of the protocol is that all nodes are synchronized and rotate in the same direction. When a node is positioned along the current angle of another node, it may send its data to this node. Depending on the density of the network, it could happen that multiple nodes belong to the same slot. The cycle in which a node is allowed to use the slot is specified by cells.

The most interesting related work to our knowledge is [19], which deals with most regular sequences. A most regular binary sequence (MRBS) is used to express the requested rates that form a rational fraction of the total available bandwidth. This results in a cyclic and deterministic sequence, which specifies for each session in which slot data should be sent in order to achieve the requested rate. However, the most regular sequences of different sessions can try to allocate the same slot, which needs to be solved by means of a conflict resolution algorithm. By means of a most regular code sequence (MRCS), it is possible to share a single slot, but the details about the allocation is neglected. The MRCS creates exactly the same sequence as the MRBS, with the exception that the ones and zeros are replaced by codes, which are a power of two, respectively, higher and lower than the requested fraction of the capacity. This result in a rate that is too fast in some cases, too slow in other cases, which leads to an average rate equal to the requested rate.

III. The algorithm

The first goal of our protocol is to create a periodic TDMA schedule at runtime. The schedule should allocate bandwidth to the sensors, such that it approximates the requested bandwidth. The periodicity of the schedule ensures that the scheduling information needs to be given only once. The second objective is to allow a regular data flow from all sensors, both from high and low bandwidth sensors. Furthermore, it is our aim that any change in the network (and thus schedule), must not have any impact on the already existing schedules. All of these goals need to be fulfilled while restricting the protocol overhead. Our solution to meet these requirements is twofold. First, we approximate the fraction of the requested bandwidth over the available bandwidth per slot, by means of an Egyptian Fraction [20,21], that is, a sum of distinct unit fractions. Second, in order to guarantee a collision free operation, every unit fraction is scheduled by means of a binary tree.

A. Methodology

In order to comply to a request for bandwidth, a sufficient number of slots needs to be allocated. The number of slots required to comply to the requested bandwidth per frame, is equal to the division of the requested bandwidth by the available bandwidth per slot. This results in an integer part and a fractional part. In order to make the most efficient use of the available bandwidth, the fractional part is approximated by means of an Egyptian Fraction, where the unit fractions have a denominator equal to a power of two. The last fraction needs to be the lowest possible unit fraction, which still is big enough such that the approximation is at least equal to the fractional part. For example, the fraction $\frac{435}{116}$ can be approximated as: $3 + \frac{1}{2} + \frac{1}{4}$.

We also represent the remaining integer part as an Egyptian Fraction, multiplied by the number of slots per frame. Thus, the resulting unit fractions need to be the representation of the integer number of slots, divided by the total number of slots per frame. It is required that the number of available slots per frame is equal to a power of two, since we are working with Egyptian Fractions that have a denominator equal to a power of two. Hence, the fraction $\frac{435}{116}$ would be approximated as: $2 + 1 + \frac{1}{2} + \frac{1}{4}$. The inverse of every unit fraction can be considered as the number of frames that determine the interval between two subsequent slots. Due to this cyclical character, it is sufficient to indicate the start position of the cycle in order to have a completely defined slot schedule. The start position of each fraction, which is defined as the offset relative to the start position of the first fraction, is obtained through a binary tree, depicted in Figure 1.

This is clarified by means of an example. The positions for each of the following fractions $\frac{1}{2} + \frac{1}{4} + \frac{1}{8} + \frac{1}{16} + \frac{1}{32}$ can be found by following the tree until its level has been reached. The most restrictive fraction, $\frac{1}{2}$, uses the resource half of the time. Thus, it can have 0 or 1 as start position. Both positions in the

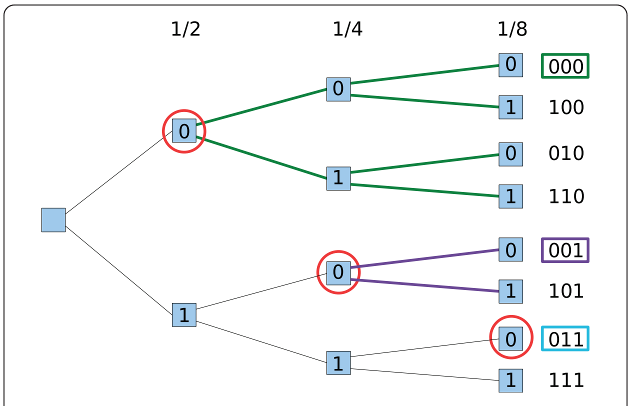

Figure 1 Binary tree.

binary tree at the level of $\frac{1}{2}$ are still free. As a rule, first the path with the 0 is followed, hence position 0 is preserved for the fraction $\frac{1}{2}$. The next unit fraction that needs to be scheduled, is the fraction $\frac{1}{4}$. Fraction $\frac{1}{2}$ already occupies position 00000 and 00010, the only remaining positions at the level $\frac{1}{4}$ are 00001 and 00011. The rule to follow first the path with a 0 leads to the reservation of position 00001 for the fraction $\frac{1}{4}$. By repeating the procedure for all unit fractions, a start position can be found for each unit fraction, such that no fraction interferes with another. The resulting allocation of the positions can be found in Figure 2.

The start positions of the fractions are determined by means of a binary tree method, but can also be expressed as a formula. Formula (1) depicts the start position, Fpos$_n$, of a fraction f_n, expressed as the offset relative to the start position of the first fraction, f_0.

$$\mathrm{Fpos}_n = \begin{cases} 0 & (n = 0) \\ \sum_{i=0}^{n-1} \frac{1}{2}\frac{1}{f_i} & (n > 0) \end{cases} \quad (1)$$

with f_i being the unit fractions. Knowing that the unit of f_i is $\frac{1}{\text{frames}}$, the start position, Fpos$_n$, is expressed as the number of frames. The Formula (1) denotes that the offset is equal to the half of the sum of all periods of previous fractions. From this can be derived that the start position of fraction f_i occurs in the middle of the period of fraction f_{i-1}.

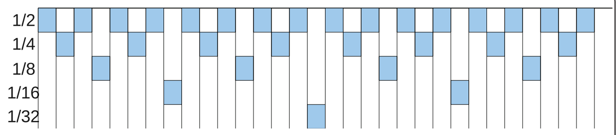

Figure 2 Binary split allocation of $\frac{1}{2}+\frac{1}{4}+\frac{1}{8}+\frac{1}{16}+\frac{1}{32}$.

B. Example

In order to illustrate the operation of the protocol, we show the resulting slot allocation of the requested bandwidth, equal to 2.75 bytes per frame, with eight slots of one byte per frame. By dividing the requested rate through the slot size, we obtain the number of slots per frame necessary to provide part of the requested bandwidth. Representing the resulting integer number as the total number of slots per frame, multiplied by an Egyptian Fraction, leads to: $8 \times \frac{1}{4} = 2$. This unit fraction $(\frac{1}{2-1})$ is positioned at slot zero of frame zero, according to the binary allocation formula, Formula (1).

Since the bandwidth of the scheduled slots is not yet sufficient to handle the requested rate, extra slots need to be scheduled. The remaining bandwidth, that needs to be scheduled, is equal to 0.75. The fraction $\frac{0.75}{1}$, which can also be written as $\frac{3}{4}$, needs to be represented as an Egyptian Fraction. The resulting series is equal to $\frac{1}{2} + \frac{1}{4}$.

The starting position for fraction $\frac{1}{2}$ is equal to: $\frac{1}{2} \times \frac{1}{2}$, according to Formula (1). Therefore, the fraction starts in slot two, since the total number of slots per frame, multiplied by the starting position, represents the starting position as the slot number, instead of the frame number. The calculation of the starting position for fraction $\frac{1}{4}$, reveals that the fraction starts at: $\frac{1}{4} + \frac{1}{2} \times \frac{1}{1/2}$, which is equal to $\frac{1}{4} + 1$. This means that fraction $\frac{1}{4}$ is scheduled, such that it uses the same slot as fraction $\frac{1}{2}$, but in different frames, the starting position of fraction $\frac{1}{4}$ is slot two in frame one.

The result is shown in Figure 3, where the requested rate is represented as: $2 + \frac{1}{2} + \frac{1}{4}$. Fraction 2 is scheduled in slots 0 and 4 in each frame. Fraction $\frac{1}{2}$ is scheduled in slot 2 in frames 0, 2, 4,... and fraction $\frac{1}{4}$ is scheduled in slot 2 in frames 1, 5, 9,....

The allocation of the slots provides a bandwidth of 11 bytes, each four frames. Converting this to the available bandwidth per frame, results in 2.75 bytes per frame, which is the requested bandwidth.

C. Discussion

We consider every requested bandwidth as being a fraction of the total available bandwidth. Due to the requirement of having a periodic slot allocation schedule, we need to find a common factor between the fractions that represent the requests of the different sensors. The gcd can be considered as such a common factor. However, calculating the gcd of all fractions, yields to a different schedule each time a new request is added. This conflicts with our requirement that an update of the schedule should not pose any conflicts

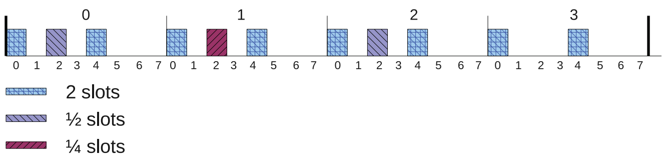

Figure 3 Allocation of 2.75 bytes per frame in a frame with a capacity of 8 bytes.

with the already existing slot allocations. As Figure 4 shows, unique unit fractions that have denominators equal to a power of two can be easily combined without resulting into conflicts. Additional unit fractions can be fitted in the remaining space, without disturbing the already allocated fractions.

The fraction of the requested bandwidth over the available bandwidth can be approximated according to such unit fraction. However, a simple approximating of the fraction leads to a large quantization error. Hence, the approximating of the requested fraction by an Egyptian Fraction, where all unit fractions have a denominator equal to a power of two. By multiplying the resulting Egyptian Fraction by the number of slots per frame, we obtain the number of slots used per frame. The remaining fractional terms indicate that a slot is scheduled once during the period determined by the fraction. This period, which is expressed in number of frames, is equal to the inverse of the fraction.

In order to prevent an infinite series that results in an unstable system, two constraints are introduced. First, the largest possible denominator is bounded in order to prevent infinite or very long sequences. Second, the total number of slots per frame needs to be a power of two, such that the fraction of the requested slots over the total number of slots can be represented perfectly by means of an Egyptian Fraction with a denominator equal to the power of two.

An interesting property of this action is that the requested bandwidth is split up into higher and lower frequency parts. A sensor gets access to the medium at least in periodic intervals equal to the highest frequency. The lowest frequency determines the cycle.

By considering the requested bandwidth as a frequency, it is possible to allocate the number of required slots once during the period of that frequency. A bandwidth request of half the bandwidth, would then have a frequency of $\frac{1}{2}$. Applying the proposed method would allocate a slot every two slots for this request, resulting in an evenly distributed allocation. This prevents a sensor from monopolizing the wireless medium, and thus obstructing other sensors.

Also, since every unit fraction of the approximation can be considered as a frequency, we only need a starting position to obtain a fully determined slot allocation schedule. The use of a binary tree guarantees that any additional fraction does not interfere with the already scheduled fractions, but it also ensures that the fractions are equally spread out over the available slots. From the Formula (1), which represents the binary tree in a mathematical form, and Figure 3, it can be noticed that the starting position of every fraction is in the middle of two successive slot schedules of the previous fraction. Hence, we obtain the interference free and equally spreaded slot allocation. The scheduling problem is thus reduced to merely following the path in a binary tree and checking whether the path is still free.

The accuracy of the Egyptian Fractions, regarding the fractional slots, depends on the smallest possible unit fraction. The lower this unit fraction, the more accurate the approximations are, but also the more frames a cycle consists of. This will be discussed in detail in the following sections.

Note that the approximation is a series of fractions of which the denominator is equal to a power of two. This information can be contained through a binary representation by representing each fraction as a single bit. Moreover, the advantage of using a periodic system, is that there is only need for the frequency and the start position. In this way, a lot of information can be given with only a few bytes of data.

For example, if the precision of the fractions is 128 (the lowest possible fraction is $\frac{1}{128}$), the sum of fractions $\frac{1}{4} + \frac{1}{32} + \frac{1}{64}$ can be expressed as $(128 \times 1/4)+(128 \times 1/32)+(128 \times 1/64)$, or simplified as 32+4+2. Putting this in a binary representation results in 0010 0110. Only a single byte is needed in order to represent the entire Egyptian Fraction. For each unit fraction, the starting position needs to be indicated, that is, the slot id and the frame in which it is scheduled. The size of the slot

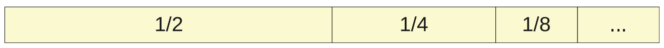

Figure 4 Binary split up.

id is determined by the number of slots within a frame, while the size of the frame information is determined by the precision of the fractions. Altogether, for a network with 8 slots per frame, and a fraction precision of 128, one byte is required to represent the fractions and 3 + 7 bits per fraction for the slot id and the frame, respectively. With a precision of 128, a maximum of 7 fractions can be achieved, which means 7 × 10 + 7 bits, which is equal to 77 bits, this is 10 bytes to send a complete assignment information, which needs to be sent only once to the requesting sensor.

D. Implementation

The aforementioned protocol is a centralized algorithm to schedule slots in a TDMA-based MAC. Therefore, the protocol can be combined with several TDMA MACs. In [1], we provided an example implementation, based upon a tree topology, where we first let the sensors synchronize to each other. Afterwards, the new sensor needs to announce its required bandwidth by sending an identification packet to its parent, which it forwards to the sink. Since this is the only moment where a collision could occur, a backoff method needs to be used, which allows a sensor to send this request again after a variable number of frames, if it has not received any acknowledgement yet. The sink uses the scheduling protocol to calculate the slots that need to be allocated and sends the slot allocation to this new sensor. As soon as the sensor receives its slot allocation, it can go into active mode, start transmitting its data and start listening to new sensors as well.

IV. Scheduling analysis: ideal data arrival rate

In order to analyze the performance of the proposed scheduling method, we simulate the scheduling on a single sensor that has a maximum bandwidth of 19,200 bits per second. Unless otherwise mentioned, all simulations make use of a frame with a duration of one second, consisting of eight slots, each having a capacity of 300 bytes and a duration of 125 ms. The bandwidth of a single slot per frame is thus 300 bytes per second.

First, an approximation of the fraction of the requested rate over the maximum capacity is made, i.e., representing the fraction as an Egyptian Fraction. The unit fractions contained in this sequence are scheduled one by one, and afterwards, an analysis is performed by simulating the progress of time concerning the sensor network operation. The simulations are performed for various bandwidths, that is, every possible integer rate in bytes per second, lower than the maximal capacity. The resulting buffer size and latency are calculated for every rate. Figure 5 depicts a flow of the scheduled outgoing data by using our scheduling algorithm and the ideal linear gradient of the data arriving at a certain rate. It

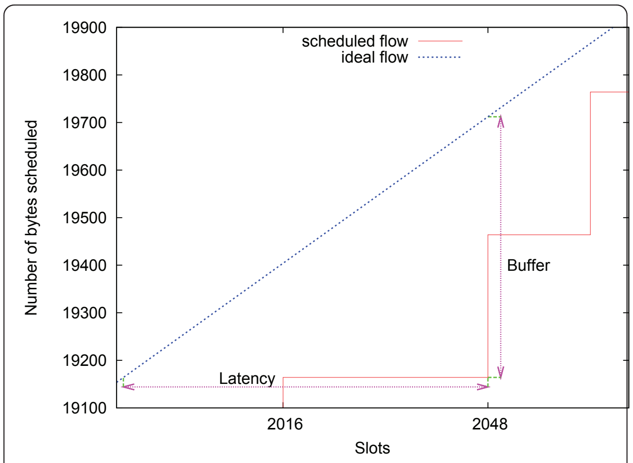

Figure 5 Latency and buffer size definitions.

illustrates the definitions of latency and buffer size. The latency is defined as the time between the ideal gradient and the flow of our protocol. If we consider the ideal flow to be the incoming data that needs to be processed by our protocol, we can say that the latency is the maximum time that the incoming data needs to wait before being processed. The buffer size can be defined as the amount of data that needs to be stored, before it can be processed. From these definitions, it is clear that the latency is in direct relation to the buffer size, i.e., latency $= \frac{\text{buffer\_size}}{\text{rate}}$. Since the analysis is more intuitive from a buffer size point of view, we first analyze the buffer size to have an idea of the latency.

A. Maximum buffer size

According to the methodology of the protocol, slots are allocated at periodic intervals, determined by a series of frequencies that approximate the requested rate. The allocations are made such that the reserved bandwidth is lower than the requested bandwidth, until a slot is scheduled according to the last frequency in the series. The last slot allocation compensates the difference between the requested bandwidth and the already allocated bandwidth.

This design can also be found in Figure 6, which depicts the ideal arrival of the data (blue dotted line) and the scheduled transmission of the data (red line) according to our scheduling protocol.

The requested arrival rate is 77 bytes per second and is approximated by $\frac{1}{4} + \frac{1}{128}$. The resulting Egyptian Fraction signifies that one slot is used for 1/4th + 1/128th of the time. According to our algorithm, slots need to be allocated to the sensor in a periodic manner. Every 4 frames, a slot needs to be allocated to the requesting sensor. And every 128 frames, one extra slot is reserved to compensate the difference between the ideal arrival rate and the previously reserved bandwidth.

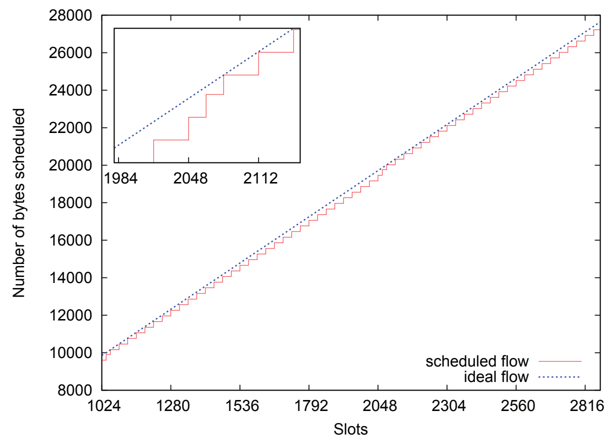

Figure 6 Approximation with 8 slots, compared to linear increasing function for 77/2400, approximated to 1/4 + 1/128.

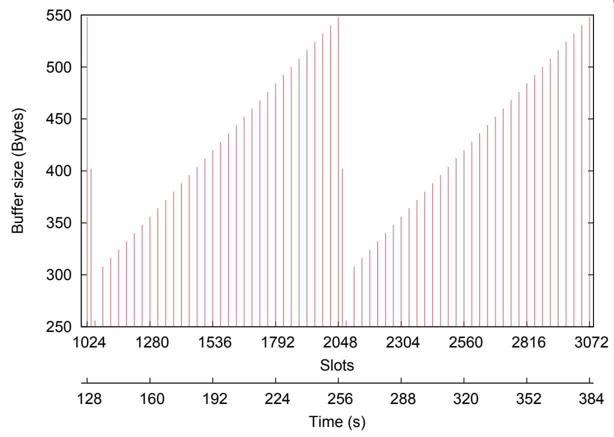

Figure 7 Buffer occupation, total amount of resources is 2400 bytes per second, the requested amount is 77, approximated to $\frac{1}{4}+\frac{1}{128}$.

Since the sum of both fractions is less than the bandwidth of a single slot per frame, the same slot is used for the whole request. By means of the binary tree, the first frame in which the slot is scheduled, can be calculated for each fraction. The slot is scheduled the first time at frame zero for fraction $\frac{1}{4}$. For fraction $\frac{1}{128}$, the slot is scheduled the first time at frame two.

Thus, the sensor can use the slot at frames 0, 4, 8, 12,... as a result of fraction $\frac{1}{4}$. And the same slot can be used by the sensor at frames 2, 130, 258,..., to realize fraction $\frac{1}{128}$.

Figure 6 shows the slotted operation, which can be noticed by the resulting step format of the scheduled data. The fact that the number of arriving data rises faster compared to the slotted transmission of the data, is a result of the scheduled slots according to a unit fraction that provides a lower bandwidth compared to the requested bandwidth. The difference between the arriving and outgoing data increases, until slot 2064 (indicated in the small figure on the top left corner of Figure 6), which is slot zero of the 258th frame. This is the slot that is scheduled according to fraction $\frac{1}{128}$ in order to compensate for the difference between the ideal arrival rate and the already reserved bandwidth.

This indicates that the protocol complies to our objective, there is a kind of periodicity in the behavior of the protocol. From the figure it can be noted that the period is 1024 slots, or 128 frames. The length of this period is determined by the number of slots and the lowest fraction which the approximation consists of. We elaborate on this item later on.

Due to the representation of the requested bandwidth as an Egyptian Fraction, which results in this periodicity, not all available data is sent immediately. This can also be seen in Figure 7, which depicts the buffer size during the different slots for the request of 77 bytes, scheduled as $\frac{1}{4}+\frac{1}{128}$. More data is arriving than being transmitted during the scheduled slots for fraction $\frac{1}{4}$. This explains why the buffer size is increasing until the slot for fraction $\frac{1}{128}$ is scheduled. Based on the results that have been shown so far, it can be expected that it is possible to calculate the maximum buffer size. Within the period of $\frac{1}{128}$, 32 slots that represent fraction $\frac{1}{4}$ are scheduled. 31 of them result in an increase of the buffer size with 8 bytes (77 × 4 - 300). Hence, the data that has not been scheduled yet is 248 bytes (8 × 31). The scheduling of that extra slot resolves the difference between fraction $\frac{1}{4}$ and the ideal arrival rate. This results in a periodic behavior of the buffer size with an interval determined by the lowest fraction, which is here $\frac{1}{128}$. The maximum buffer size is obtained when the last slot is scheduled that is not according to the last unit fraction, i.e., it is the last slot before the scheduling of a slot according to the last unit fraction. Since at that moment a complete slot is yet to be transmitted, the maximum buffer size is equal to the calculated amount of data that has not been scheduled yet, increased by the size of a slot. By adding the 300 bytes of the slot size, we get a maximal buffer size of 548 bytes. If we compare this to the figure, we see that this calculation is a perfect match to the obtained maximum buffer size.

To generalize, we claim that by means of Formula (2), the maximum buffer size can be calculated, based upon the following parameters: the requested amount, the slot size and the Egyptian Fraction that approximates the requested amount.

$$\mathrm{Max\_buff}_k = R\frac{1}{f_0} + \sum_{i=1}^{k}\left(\frac{f_{i-1}}{f_i} - 2\right)\frac{1}{f_{i-1}}\left(R - \sum_{j=0}^{i-1} f_j S\right) \quad (2)$$

with R being the requested bandwidth (bytes per frame), $f_0 \dots f_n$ being the unit fractions that form the approximation and S (bytes) representing the slot size. The proof can be found in Appendices A, B, and C.

As an example, we calculate the maximum buffer size, according Formula (2) for the request of 77 bytes per second, approximated as $\frac{1}{4} + \frac{1}{128}$:

$$\begin{aligned} \text{Max\_buffer} &= 4 \times 77 + \left(\frac{128}{4} - 2\right) \times 4\left(77 - \frac{300}{4}\right) \\ &= 308 + 30 \times 4 \times 2 \\ &= 548 \end{aligned} \quad (3)$$

The formula matches with the measured result. The fact that the formula depends on the requested rate and the Egyptian Fraction gives reason to investigate the relation between the maximum buffer size and the requested rate. Figure 8 depicts the measured maximum buffer size for all the possible integer rates that can be requested from a resource with a maximal capacity of 2400 bytes per second and 8 slots per frame. At first sight, the maximum buffer size seems to behave randomly in function of the requested bandwidth, but there is a pattern. A more detailed view of the figure reveals this.

Figure 9 zooms in on the section with requested bandwidths between 150 and 230 bytes per second. The full red lines in the figure indicate the maximum buffer size needed for that request. The dotted blue lines form the binary representation of the unit fractions that appear in the approximation of the requested amount. The top blue line is the smallest fraction ($\frac{1}{128}$), the next blue line is the double of the fraction of the previous line and so on, until we reach the bottom blue line, that is fraction $\frac{1}{2}$. Notice that the approximation of the requested bandwidth of 185

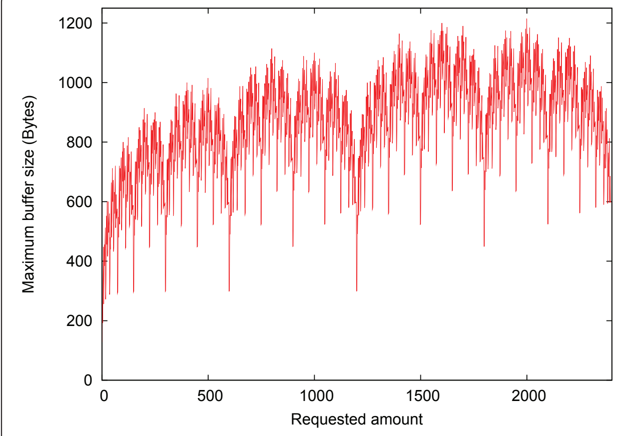

Figure 8 Measured maximal buffer occupation, 8 slots.

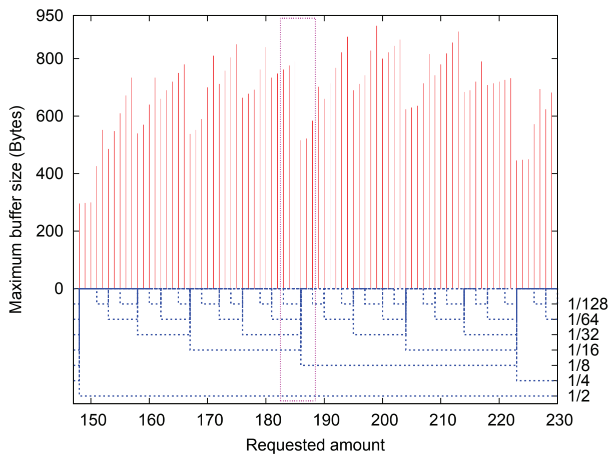

Figure 9 Measured maximal buffer occupation, 8 slots and approximation, rates 150 till 230 bytes per second.

bytes per second can be deducted from this figure, it is represented as $\frac{1}{2} + \frac{1}{16} + \frac{1}{32} + \frac{1}{64} + \frac{1}{128}$. The approximation of a requested bandwidth of 186 bytes per second is $\frac{1}{2} + \frac{1}{8}$.

From this figure can be noticed that the more fractions are used to approximate the requested amount, the higher the maximum buffer size is. The maximum buffer size increases in a more steep gradient if an extra fraction is added to the approximation. Another phenomenon is that if a single larger unit fraction is used, instead of using a series of unit fractions, the maximum buffer size decreases. This observation can be made for example if we compare rate 185 (approximated as $\frac{1}{2} + \frac{1}{16} + \frac{1}{32} + \frac{1}{64} + \frac{1}{128}$) and rate 186 (approximated as $\frac{1}{2} + \frac{1}{8}$), indicated by the dotted pink rectangle in the figure. These observations point out that every unit fraction adds its own surplus to the maximum buffer size.

In summary, the formula and the figures indicate that each fraction in the approximation adds a certain surplus to the maximum buffer size. Therefore, in order to decrease the maximum buffer size, the number of unit fractions within an Egyptian Fraction could be constrained. On the other hand, this results in a higher waste of the available resources since the approximation is not a tight match, hence, the bandwidth usage efficiency drops. Figure 10 illustrates the effect of limiting the number of fractions. In the figure, the smallest fraction used is $\frac{1}{16}$ (instead of $\frac{1}{128}$ in Figure 8). It can be noted that limiting the number of fractions results in a large decrease of the maximum buffer size, but also the fine granulation has been lost. This signifies that the approximations are not so precise anymore and a lot of capacity is wasted in favor of the buffer size.

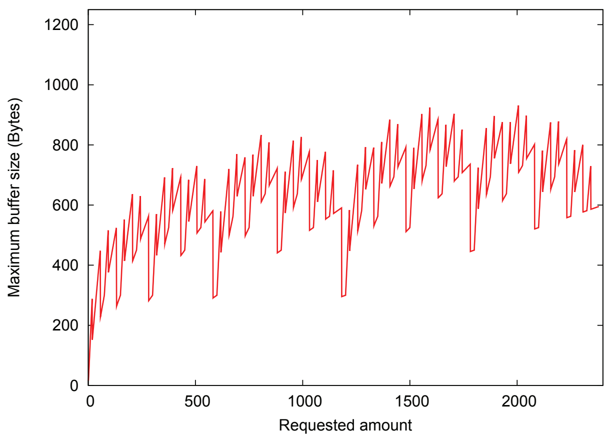

Figure 10 Maximal buffer occupation, 8 slots, fractions limited to 1/16.

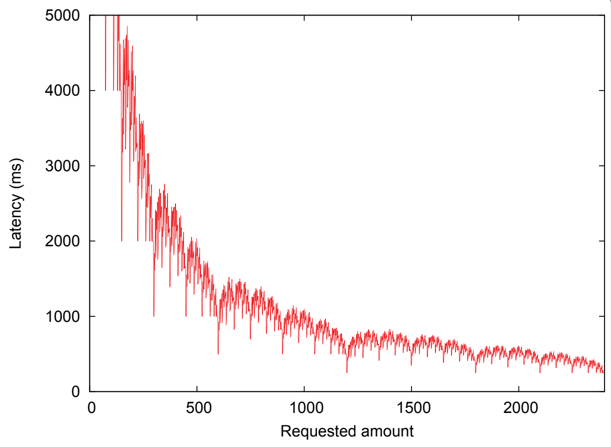

Figure 12 Maximal latency, 8 slots.

B. Maximum latency

As mentioned before, there is a direct relation between the latency and the buffer size. This can be noticed when comparing the gradient of the buffer size (Figure 7) to the gradient of the latency (Figure 11). The maximum latency can be considered as the time required to receive a number of bytes, equal to the maximum buffer size, at the requested rate (the maximum buffer size divided by the requested amount equals the maximum latency). For example, The maximum buffer size of the fraction $\frac{77}{2400}$ is 548 bytes. The time needed to receive this data at a rate of 77 bytes per second is 7.11688 s (548 bytes/77 bytes per second). Converted to milliseconds, this gives 7116.88 ms, which can be verified in Figure 11.

Intuitively, we can predict that the latency for the smaller requested amounts will be higher than for the larger amounts. Requests that are smaller than the size of a single slot need to share the resource with other sensors. They get access to the resource once every x frames, and need to wait relatively long, because they need to gather enough data to send a large quantity at once.

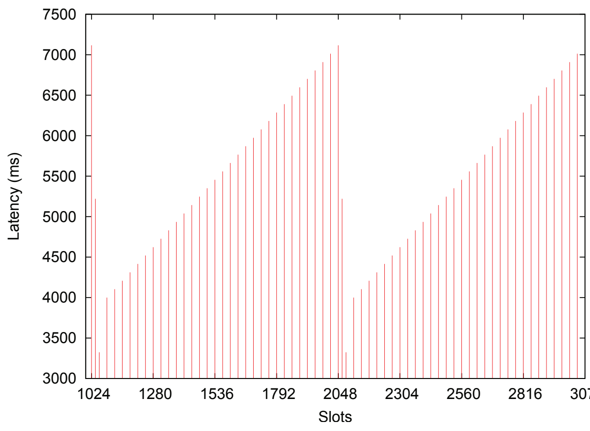

Figure 11 Latency of a requested rate 77, approximated to $\frac{1}{4}+\frac{1}{128}$.

Figure 12 shows the maximum latency that occurs for each integer requested bandwidth, with a maximum capacity of 2400 bytes per second and 8 slots per frame. We see that the smaller amounts indeed have a larger latency. The highest maximum latency can be found at the requested rate of one and two bytes per second. Both have a latency of 128,000 ms. However, the latency has a quadratic gradient and a rather low latency is quite fast achieved for the requested amounts. The best latency that can be noticed is 250 ms, which is two times the slot duration. This is to be expected, because the highest possible frequency, that can be obtained, is half the number of slots in a frame. Slots are scheduled according to this frequency in an interleaving manner, thus at least every two slots data is sent. The result is a minimum latency equal to two times the duration of a slot.

Since the maximum latency can be calculated from the maximum buffer size, there should be a similar pattern in the gradient of the latency as the one that can be seen at the gradient of the maximum buffer size, caused by the sequence of fractions that an approximation consists of. In Figure 13, being a small section of the previous figure, it can be seen that although the gradient of the latency is descending, it still shows a similar behavior as the maximum buffer. The full red lines indicate the maximum latency for that request, the dotted blue lines are the binary representation of the unit fractions that the approximation of the requested rate consists of, similar as in Figure 9.

The more fractions are used to approximate the requested amount, the higher the latency is. However,

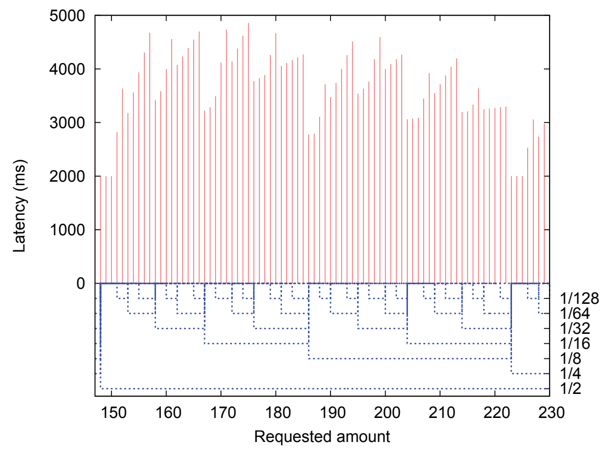

Figure 13 Maximal latency and the approximation composition, 8 slots.

we need also to take the data rate into account, which is a linear increasing function. We can still see the small inclinations when an extra fraction is added to the approximation, such as we can see with the maximum buffer size. But the rate has a big influence on the equation, such that the result is the quadratic gradient. For example, the top maximum buffer size is at the request of 1999 bytes per second (with 8 slots), while the maximum latency is small at that instance. This is because of the high data rate of the request. It does not need to take more time to fill a large buffer at a high data rate than a smaller buffer at a lower data rate.

C. Latency control

As a consequence of the relation between the latency and the buffer size, and due to the fact that the maximum buffer size can be controlled, it is possible to control the maximum latency. As previously discussed in Section IV-A, the maximum buffer size can be lowered by limiting the maximum number of fractions that an approximation can consist of. The second parameter that, according to Formula (2), has an influence on the maximum buffer size, and hence the maximum latency, is the slot size.

Up till now, we only discussed the results of a virtual sensor that has a frame with a duration of one second and is split into 8 slots, that is, a slot size of 300 bytes. For sensors that do not need to send a lot of data, this means that they have to wait a long time before they have gathered enough data to send. Although it is possible to schedule such low rate sensors in the network, a more efficient approach is possible. By increasing the number of slots per frame, while keeping the frame length constant, leads to a decrease in slot slot size, hence, a decrease of the maximum latency.

Figure 14 depicts the maximum buffer size for all possible requests by using 16 slots during an equally sized frame. We can notice that the figure has a similar gradient as Figure 8, but with a lower maximum buffer size. The requested rates, that needed a half slot previously by scheduling a single slot every two frames, are served by one full slot now that is scheduled every frame. So it is obvious that the sensors need to store twice less than with the bigger slot size. Since a sensor that needed to wait every two frames to send data, can send every frame now, it is clear that the data is being forwarded faster, hence, the latency should be lower. This can be seen in Figure 15, where the maximum latency is depicted for a frame that has been split in 16 slots. The minimum latency, 125 ms, is two times the duration of a slot.

We notice that, in theory, increasing the number of slots leads to a lower buffer size and latency. However, in practice, additional information needs to be sent together with the data. This information can be about the source of the data, the type or amount of data, perhaps a Cyclic Redundancy Check (CRC) so that we are able to verify whether the data is not corrupted while it was transferred from one sensor to another. Furthermore, when sending data, the physical layer adds some hardware dependent bits to the data packet. The size of this information is independent of the amount of data, hence, the smaller the slots are, the less efficient the data throughput.

V. Scheduling Analysis: bursty data arrival

In the previous section, we analyzed our proposed scheduling protocol by considering a constant data stream as the input data. We analyzed what kind of influence has the requested rate upon the resulting buffer size and latency. Since we use data slots of a certain size and we calculated the most optimal time to send data at that

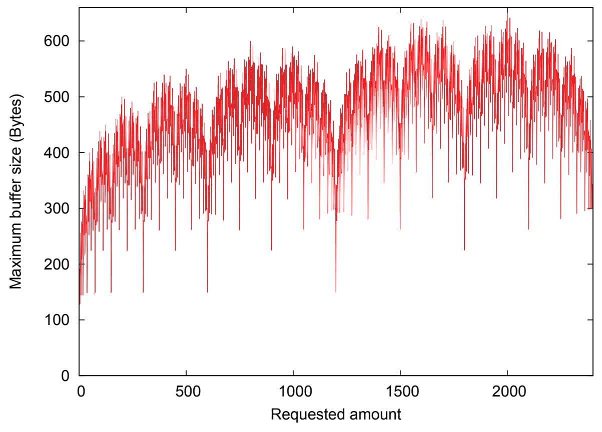

Figure 14 Maximal buffer occupation, 16 slots.

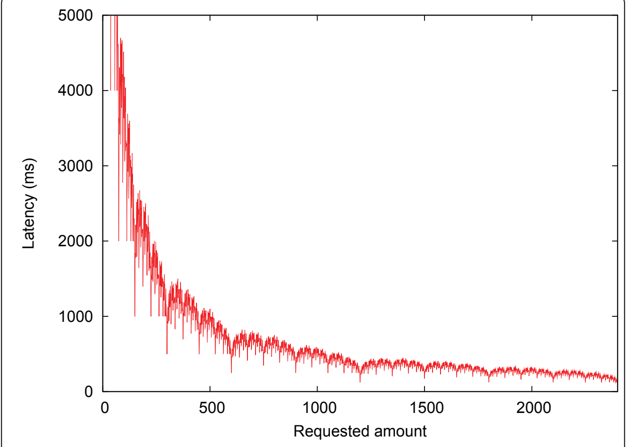

Figure 15 Maximal latency, 16 slots.

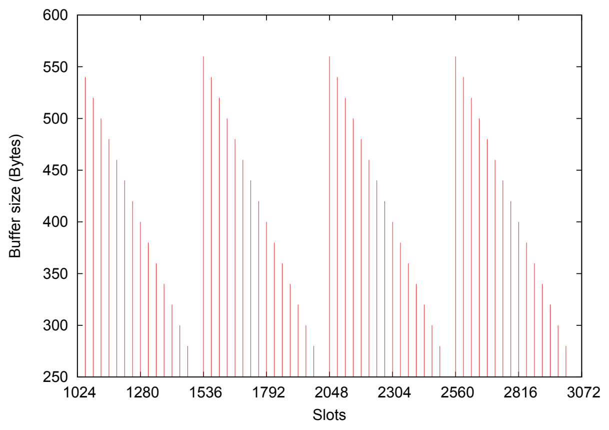

Figure 16 Buffer occupation, packet size 280 bytes, Total amount of resources is 2400 bytes per second, the requested amount is 70, approximated to $\frac{1}{4}+\frac{1}{128}$.

specific rate, we can imagine that if the data is arriving in bursts, this would have a negative effect on the performance of the protocol. In this section, we analyze the influence of a bursty data arrival.

We simulate bursts with packet sizes from 1 to 2400 bytes for each rate, ranging from 1 to 2400 bytes per second. The maximum rate of the virtual sensor is 2400 bytes per second, the frame is split up into 8 slots, and the frame length is set to one second, resulting in a slot size of 300 bytes.

A. Buffer size

Unlike the case of a continuous data arrival, when data arrives in bursts, it can happen that a slot is scheduled to process data, but there is no data available. This is the worst case scenario. The general behavior is that not as much data can be processed at the scheduled times as expected, because the data arrives in bursts, which does not need to match with the scheduled slots.

In the previous section, we found that the buffer size increases and decreases in a periodic cycle. The period of this cycle, also called the scheduling period, is determined by the lowest fraction that the approximation holds. If this period contains an integer number of arrival times of packets, it is equal to the scheduling period. This feature can be seen in Figure 16. It depicts the gradient of the buffer size at a request of 70 bytes per second and a data arrival in bursts of 280 bytes.

On the other hand, if the scheduling period is not divisible by the arrival times of the packets, we get a sequence of scheduling periods that form a cycle on their own. The number of scheduling periods that this cycle will have, can be calculated. The arrival time of the packets can be expressed by: arr_time = P/R where P is the size of the packet and R the requested rate. The demand that the scheduling period needs to be divisible by the arrival time can be expressed by: modulo(B, arr_time) = 0, where B is the scheduling period. Another way to represent the demand that the scheduling period is divisible by the arrival time is:

$$\begin{aligned}
&B|\text{arr\_time}\\
&\quad\Updownarrow \text{arr\_time} = \frac{P}{R}\\
&B \times R|P\\
&\quad\Updownarrow P = x \times \gcd(R,P)\\
&B \times R|(x \times \gcd(R,P)\\
&\quad\Updownarrow x = y \times \gcd(B,x)\\
&B \times R|(y \times \gcd(B,x) \times \gcd(R,P))
\end{aligned}$$

We can say that the scheduling period is divisible by the arrival time, if y equals to one. Even more, we can say that y represents the number of times the scheduling period needs to be repeated, before we start the periodic cycle again. In order to calculate the length of the period, we can use the following formula:

$$\begin{aligned}
&P = y \times \gcd(B,x) \times \gcd(R,P)\\
&\qquad\Updownarrow x = \frac{P}{\gcd(R,P)}\\
&y = \frac{1}{\gcd\left(B, \dfrac{P}{\gcd(R,P)}\right)} \times \frac{P}{\gcd(R,P)}
\end{aligned} \tag{4}$$

We can verify this formula, by observing Figures 16 and 17, where results are depicted from simulations with 8 slots of 300 bytes, a packet size of 280 bytes and with a requested rate of 70 and 77 bytes per second, respectively. According to the calculations, the period should be one for a rate of 70 bytes per second (gcd(70,

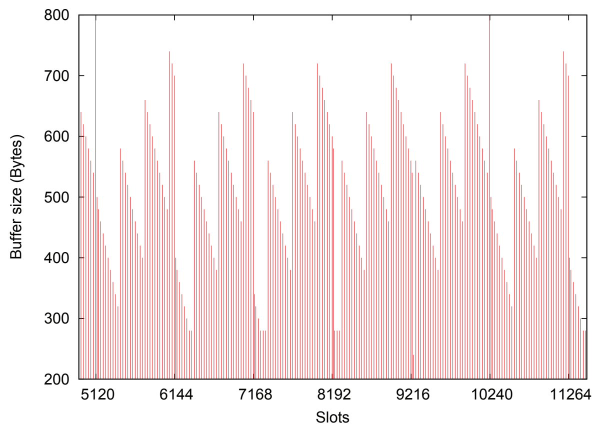

Figure 17 Buffer occupation, packet size 280 bytes, Total amount of resources is 2400 bytes per second, the requested amount is 77, approximated to 1/4 + 1/128.

280) = 70). It is clear that there is only one period during the cycle. In Figure 17, for 77 bytes per second, we can see that the scheduling period is equal to 1024 slots, while the periodic gradient of the buffer size has become 5120 slots, this is five times the scheduling period. If we calculate after how many scheduling periods the buffer size starts a new cycle, we obtain five, as it should be (gcd(280, 77) = 7 and gcd(128, 40) = 8, so the period is $\frac{280}{56}$ = 5).

Figure 17 shows nicely that the packets arrive in bursts (the sudden increase in buffer size), after which the buffer size decreases gradually at each scheduled slot. As the packet arrivals and the scheduling period do not coincide, it can happen that a packet arrives right before the scheduling period ended. At the end of the scheduling period, the fraction is scheduled that should compensate the difference between the sum of all other fractions and the requested rate. Since the packet arrives right before this fraction is scheduled, the maximum buffer size increases, compared to the case where the data arrives gradually.

Figure 18 depicts a 3D plot that represents the maximum buffer size versus the requested rate and the packet size for a simulation with 8 slots, rate from 1 to 2400 bytes per second, which is the maximum capacity and packet sizes from 1 to 2400 bytes. A first conclusion is that the larger the packet size, the higher the maximum buffer size becomes.

The data in the figure is plotted every 50 lines to enhance the visibility. Hence, a lot of the resolution has been lost. To visualize the most optimal maximum buffer sizes, Figure 19 provides a general overview of the results that represent the lowest maximum buffer sizes, shown by the dark lines. The darker the line is, the lower the maximum buffer size. It can be noticed that

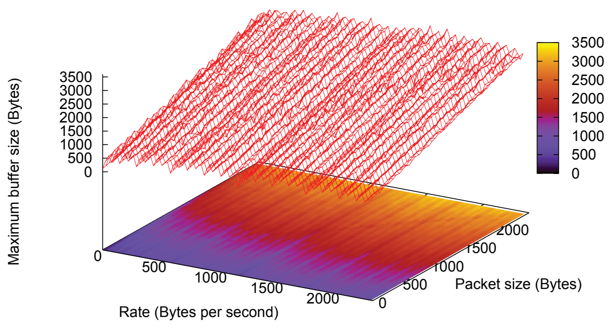

Figure 18 Maximum buffer size versus the rate and the packet size.

there are darker lines in the horizontal, vertical and diagonal direction. The maximum buffer size increases with an increasing packet size and increasing requested rate.

The gradient of the maximum buffer size, as seen in the previous section, can be recognized by the vertical darker lines. They indicate the rates where the maximum buffer size is lower in comparison with other rates at the same packet size. This indicates that when working with data arrival in bursts, the maximum buffer size is lower when limiting the number of fractions of the approximation.

The horizontal darker lines, on the other hand, are the result of the relation of the packet size to the slot size. In this simulation, we used slot sizes of 300 bytes. At packet sizes that are a multiple of 300 bytes, there are darker horizontal lines. This means that when the data arrives, at a certain rate, in bursts of this packet size, it requires a smaller maximum buffer size than with another packet size. Intuitively, something like this could be expected. If a packet size is a multiple of the

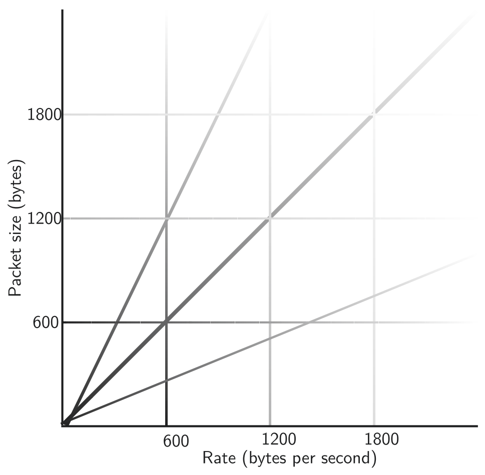

Figure 19 Lowest maximum buffer size lines.

slot size, we expect that this has a positive effect on the maximum buffer size.

To get a deeper understanding, we plot the maximum buffer size over the rate for two different packet sizes, 1190 bytes and 1200 bytes, in Figure 20. The packet size of 1200 is one of those horizontal darker lines, thus the maximum buffer size should be lower there. We can see that the packet size of 1190 bytes has a similar flow as with the gradual arrival of packets, but with an offset that is the result of the packetized arrival of the data. The flow of the packet size of 1200 bytes shows a maximum buffer size that looks quantized. This is actually the effect of the packet size being a multiple of the slot size. This effect is also noticeable if the slot size is divisible by the packet size.

This is even more clearly depicted in Figure 21, which shows the maximum buffer size at a fixed rate of 9 bytes per second for each simulated packet size. Naturally, as we could notice in the 3D figure, the data that arrives in bursts causes a higher buffer usage. Here we can see that at certain packet sizes, the maximum buffer size is lower. Interesting to note is that when using packet sizes between 0 and 300 bytes, for which 300 bytes forms a multiple, all have the same maximum buffer size. The same counts for packet sizes between 300 and 600 bytes, and further. There is some kind of repetition, determined by the slot size (which is here 300 bytes), which is to be expected, since the slot size of 300 bytes determines the packet sizes at which a lower buffer size can be detected. We notice that the maximum buffer size is not a nice linear function, but we can determine an upper bound, namely the value we calculated by means of the gradual arrival of the data, increased by the packet size. If we look at Figure 22, where the lowest maximum buffer size equals 300 bytes, we notice that this upper bound function also holds for this rate.

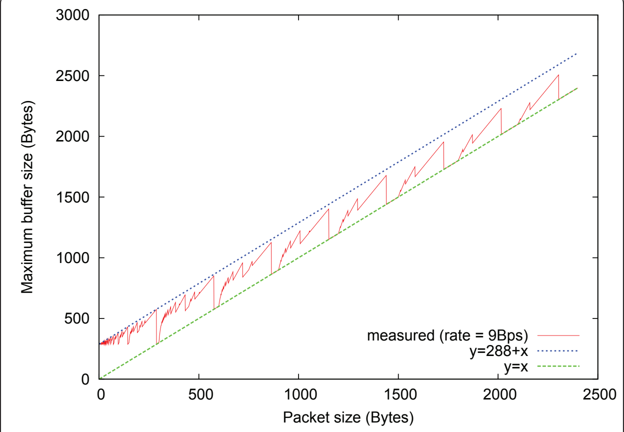

Figure 21 **Maximum buffer size versus the packet size at 9 bytes per second**.

The only lines we did not explain yet are the diagonal darker lines. The fact that they are diagonal, means that for a certain rate-packet size combination, they form a minimum. The cyclic character of the scheduling period is the reason for this phenomenon. The number of scheduling periods that form a cycle is defined by Formula (4). The number of periods depends on the packet size, the rate and the scheduling period. This is our rate-packet size relation. To give an example, one of the packet sizes for which y of Formula (4) equals one, is 600 bytes (gcd(1200, 600) = 600 and gcd(2, 2) = 2). One of the packet sizes for which y equals one at a rate of 1190 bytes per second, is 595 bytes (gcd(1190, 595) = 595 and gcd(64, 2) = 2). This gives our diagonal darker line. Notice that there are many packet sizes that fulfill the requirement, so there are more than one darker diagonal line.

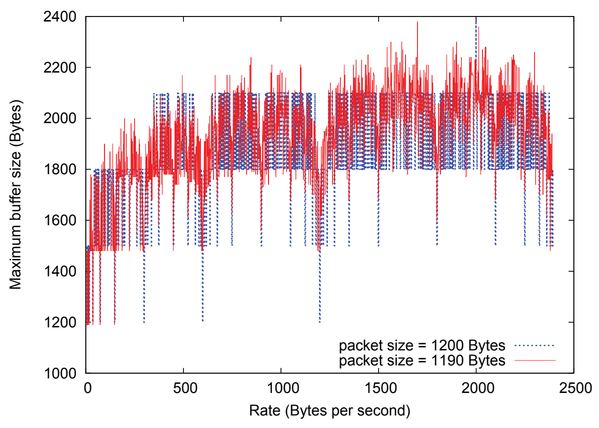

Figure 20 **Maximum buffer size versus the rate with packet sizes of 1190 and 1200 bytes**.

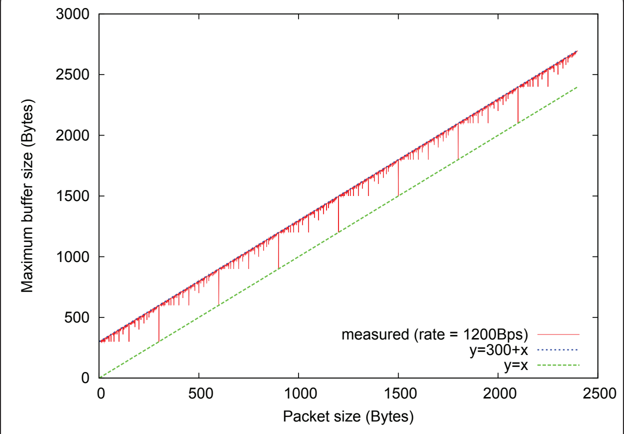

Figure 22 **Maximum buffer size versus the packet size at 1200 bytes per second**.

B. Latency

Based on the relation between the maximum latency and maximum buffer size, we expect that the maximum latency behaves like the buffer size when the data arrives in bursts. Figure 23 depicts the logarithmic function of the maximum latency over the requested rate and the packet size. We plotted the latency on a logarithmic axis in order to reveal more details, since the maximum latency of the lower rates is a lot higher than the maximum latency of the rest of the rates. The reason for this characteristic, that we already noticed by means of the gradual arrival of the data, is rather trivial. At lower rates, it takes more time to fill the transmit buffer, hence the slots are not so frequently scheduled, resulting in a high maximum latency.

The second characteristic that can be noticed, is that the maximum latency increases as the packet size increases, which is the same behavior as the maximum buffer size. There is a difference though, the packet size has a bigger influence at the lower requested rates, compared to the higher rates. The most optimal minimum latency points can also be found on horizontal, vertical and diagonal lines. Since the latency can be derived directly from the maximum buffer size, the reason why at these points a minimum latency can be found is the same.

If we compare the maximum latency for two different packet rates (Figure 24), we notice a similar behavior as with the gradual arrival of the data. The number of fractions that form the approximation plays again a factor in the determination of the maximum latency, the less fractions, the lower the latency. We can also see here that the packet size is less important at high rates, compared to lower rates.

The horizontal lines, which are the effect of the packet sizes that happen to be multiple of the slot sizes, can be seen in Figure 25. The maximum latency versus the packet size is depicted for rates of 600 bytes per second and 1200 bytes per second. Notice that at packet sizes that are a multiple of the slot size, the maximum latency reaches a minimum with packet sizes between 0 and 300, hence lower than the slot size. This phenomenon repeats itself each 300 bytes. In the figure, it can also be seen that the influence of the packetized arrival of the data is bigger at low rates than it is at higher rates.

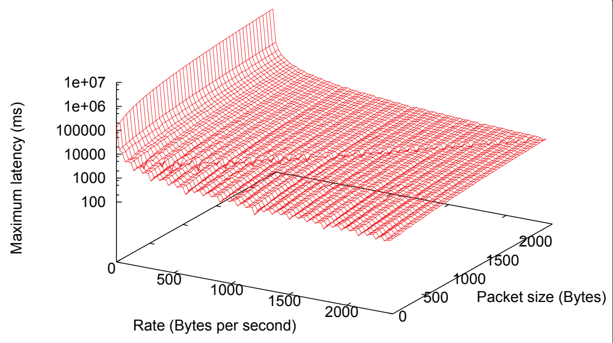

Figure 23 Maximum latency versus the rate and the packet size, 3D plot.

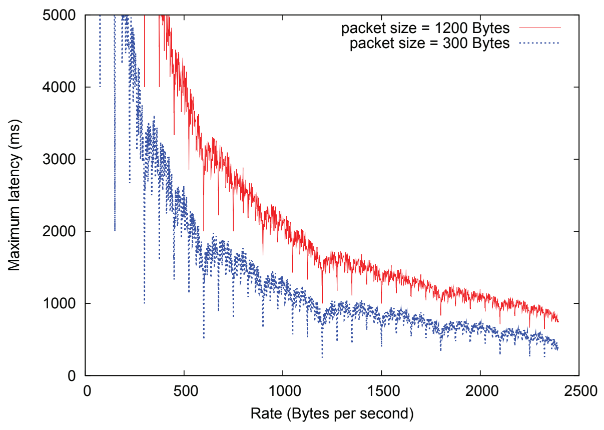

Figure 24 Maximum latency versus the rate with packet sizes of 300 and 1200 bytes.

VI. Conclusion

In this article, we propose a slotted scheduling protocol, aiming at energy preservation, real-time properties, fairness and a periodic schedule. It is designed for energy preservation. It is better to use the network for a small unit of time and then utilizing the full bandwidth, than sending each time just a bit of data. This is also more efficient towards the overhead, generated by the physical layer. The more data that is being sent at a time, the smaller the header is in comparison to the amount of data. The second goal is real-time behavior. When dealing with time sensitive material, it is imperative to receive the required data in time, otherwise the data

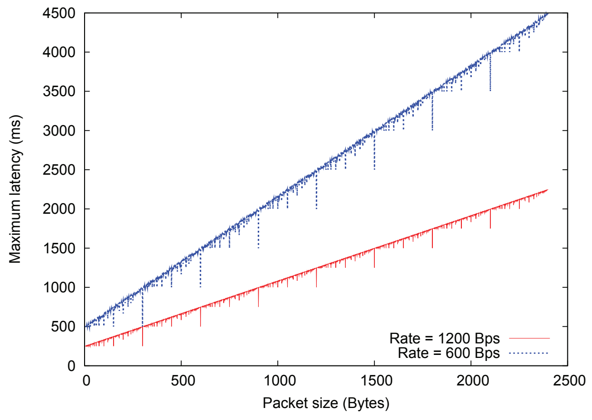

Figure 25 Maximum latency versus the packet size at 600 bytes per second and 1200 bytes per second.

could be worthless. A third objective of this protocol is fairness. Even when the network is crowded with high bandwidth sensors, the low bandwidth sensors should still be able to send their data. The protocol also needs to consider the feasibility to implement it on an actual hardware platform. Last, it needs to take into account that it can happen that a control packet gets lost.

The protocol takes several limitations into account in order to comply to all previous demands. First of all, it is designed in such a way that the slot allocation needs to be sent only once during the whole lifetime of the network. The design is based on a periodic cycle, which allows to send the slot allocation, which is afterwards repeated over and over again. Any subsequent changes in the topology also do not influence the already scheduled slots. To inform sensors about the slot allocations, only a small number of bytes is required.

We require every sensor to indicate how much bandwidth it needs. The bandwidth request is split into the number of full slots and a fractional representation of a slot. The fraction of the number of full slots over the total number of slots in a frame is represented by means of Egyptian Fractions, where the denominators are a power of two. By means of a binary tree, each unit fraction is fit into a certain slot. The selection of slots ensures that collisions are avoided. The fractional representation of a slot, which is also an Egyptian Fraction, is scheduled by means of a binary tree. The resulting allocation indicates in which frame the slot may be used. This scheme leads to a schedule that is cyclic, which cycle length is determined by the lowest fraction in the approximations. As can be noticed, this protocol is not work conserving, as the time that no slots are scheduled, can be used to put the sensors to sleep.

Because of its determined schedule, this protocol has real-time properties. We can calculate the maximum required buffer size a sensor needs for a given bandwidth. This gives the possibility to calculate the maximum latency, thanks to the direct relation between the buffer size and the latency. Therefore, the scheduling protocol is capable to schedule real-time tasks.

The analysis shows that the maximum buffer size, and hence the maximum latency, depends on the number of fractions an approximation is made of. A lower number of fractions results in a lower latency. In order to tweak the protocol for certain operating requirements, the lowest fraction that is contained in an approximation can be tuned. This results in a network that has a lower latency, but where the bandwidth efficiency is lower, some of the bandwidth is wasted to ensure the timely arrival of the data. The nice property of this tuning, is that it can be done on a per node basis. It is for example possible to have a node which is aiming at as low as possible latency, by limiting the lowest possible fraction to approximate. And at the same time, having a node that is aiming at as much as possible bandwidth efficiency by using as much fractions as possible in its approximation.

Another way to limit the latency, is the number of slots that a frame is split into. By using more slots in a frame, the frame is split into smaller pieces, hence at the same rate, the sensors use the slots two times faster. Getting an allocation that is two times faster, means also a latency that is lower. However, this manner of tuning is global, the whole network needs to have an equal amount of slots per frame.

We also noticed that the latency is quite high when the requesting rate is low. This is perfect if the goal is to preserve energy. If the goal is to have a latency that is as low as possible, then, either the lowest possible fraction needs to be limited, or the total capacity of the network needs to be downsized.

During our simulations with bursty data arrivals, we noticed that it is interesting to have an approximation that has only one or two unit fractions. It is also interesting if the slot size is divisible by the packet size. It is not so interesting to have a packet size that is bigger than the slot size. And it is also good to have a certain rate-packet size relation, according to Formula 4. These three cases give the lowest possible buffer sizes. These conclusions are also valid for the latency, but there the rate plays also a very important role: the higher the rate, the lower the latency.

Appendix A
Proof of the remaining bandwidth formula
The requested rate divided by the slot size is approximated through a series of unit fractions with a denominator equal to a power of two:

$$\begin{aligned} \frac{R}{S} &= \sum_{i=0}^{m} \frac{1}{2^{n_i}} \quad (n_i \in \mathbb{Z} \text{ and } n_i < n_{i+1}) \\ &\text{or} \\ \frac{R}{S} &= \sum_{i=0}^{m} f_i \quad (\text{with } f_i = \frac{1}{2^{n_i}} \text{ and } f_i > f_{i+1}) \end{aligned} \tag{5}$$

with R being the requested rate and S representing the slot size. R is expressed as bytes per frame, while S is expressed as number of bytes. Therefore, the unit of the sum of fractions is $\frac{1}{\text{frames}}$, that is, every fraction signifies a frequency at which slots are scheduled. $\frac{1}{f_i}$ denotes the interval, expressed in number of frames, between successive slot schedules according to fraction f_i. Every fraction, except the last one, results in the transmission of part of the data. The data that has arrived during the slot scheduling interval, determined by the specific fraction, is equal to:

$$R_i \frac{1}{f_i}$$

with R_i being the remaining bandwidth per frame (bytes per frame) and f_i being the specific fraction. A special case is the first fraction, f_0, since the remaining bandwidth, R_i, is there equal to R, the requested rate.

The scheduling of a slot, according to fraction f_i, results in the transmission of S bytes, with S indicating the slot size. In other words, the remaining bandwidth after $\frac{1}{f_i}$ frames equals:

$$R_i \frac{1}{f_i} - S$$

or expressed as the remaining bandwidth per frame, we get:

$$R_{i+1} = f_i \left(R_i \frac{1}{f_i} - S \right) \tag{6}$$

We claim that the amount of data arriving during the slot scheduling interval of a fraction f_i at a rate of R_i, the remaining bandwidth, is not larger than two times the slot size. Thus, our claim is that:

$$R_i < 2 f_i S \quad \text{or} \quad \frac{R_i}{f_i} < 2S \tag{7}$$

with R_i being the remaining bandwidth per frame (bytes per frame), that is, the bandwidth that is not covered yet by the fractions that represent a higher frequency. S denotes the slot size and $f_i = \frac{1}{2^{n_i}}$, indicating the unit fraction.

Proof:

We prove this statement through induction.

Step 1:

Since it is a requirement to have unit fractions and $n_i < n_{i+1}$, we can state that the following expression is true:

$$\sum_{i=1}^{k} \frac{1}{2^{n_i}} < \frac{1}{2^{n_0}} \quad (n \in \mathbb{Z} \text{ and } n_i < n_{i+1}) \tag{8}$$

The combination of Formula (5) and Formula (8), gives:

$$\begin{gathered} \frac{R}{S} < 2\frac{1}{2^{n_0}} \\ \Updownarrow \\ R\frac{1}{f_0} < 2S \qquad (\text{with } f_0 = \frac{1}{2^{n_0}}) \end{gathered} \tag{9}$$

The remaining bandwidth, R_0, is equal to the requested bandwidth, R, since f_0 is the first fraction.

Step 2:

We consider the following formula to be true for fraction f_k:

$$\frac{R_k}{f_k} < 2S \tag{10}$$

We now prove that Formula (7) also counts for f_{k+1}.

It is known that the amount of data arriving during the slot scheduling interval of a fraction f_{k+1} at a rate of R_{k+1}, is equal to:

$$\frac{R_{k+1}}{f_{k+1}}$$

From 6, we know that:

$$R_{k+1} = f_k \left(R_k \frac{1}{f_k} - S \right)$$

The resulting amount of data arriving during the interval determined by fraction f_{k+1} is:

$$\frac{1}{f_{k+1}} f_k \left(R_k \frac{1}{f_k} - S \right)$$

which we can also write as:

$$\frac{f_k}{f_{k+1}} \left(\frac{1}{f_k} R_k - S \right)$$

It is known that, according to Formula (6), R_k equals to:

$$R_k = f_{k-1} \left(\frac{R_{k-1}}{f_{k-1}} - S \right)$$

The substitution of R_k gives the following expression as the amount of data arriving during the interval determined by fraction f_{k+1}:

$$\frac{f_k}{f_{k+1}} \left(\frac{1}{f_k} \left(f_{k-1} \left(\frac{R_{k-1}}{f_{k-1}} - S \right) \right) - S \right)$$

Substituting all R_i, with $0 < i < k$ results:

$$\frac{f_k}{f_{k+1}} \left(\frac{f_{k-1}}{f_k} \left(\cdots \frac{f_0}{f_1} \left(\frac{R_0}{f_0} - S \right) - S \cdots \right) - S \right)$$

Since R_0 equals to R, according to Formula (9):

$$\frac{f_k}{f_{k+1}} \left(\frac{f_{k-1}}{f_k} \left(\cdots \frac{f_0}{f_1} \left(\frac{R}{f_0} - S \right) - S \cdots \right) - S \right)$$

Rewriting gives us:

$$\frac{1}{f_{k+1}} (R - f_k S - f_{k-1} S \cdots - f_1 S - f_0 S)$$

which is equal to:

$$\frac{1}{f_{k+1}} \left(R - \sum_{i=0}^{k} f_i S \right) \tag{11}$$

From Formula (5), we know that:

$$\frac{R}{S} = \sum_{i=0}^{m} f_i$$

which can also be formulated as:

$$\frac{R}{S} - \sum_{i=0}^{k} f_i = \sum_{i=k+1}^{m} f_i \quad (\text{with } 0 < k < m) \tag{12}$$

Since f_i equals to $\frac{1}{2^{n_i}}$, with $n < n + 1$, we know that:

$$f_{k+1} > \sum_{i=k+2}^{m} f_i \tag{13}$$

The combination of Formula (12) and Formula (13) gives:

$$\frac{R}{S} - \sum_{i=0}^{k} f_i < 2f_{k+1}$$

which can also be written as:

$$\frac{1}{f_{k+1}} \left(R - \sum_{i=0}^{k} f_i S \right) < 2S \tag{14}$$

From Formula (11) and Formula (14), we can conclude that:

$$\frac{R_{k+1}}{f_{k+1}} < 2S$$

Q.E.D.

Appendix B

The start position of a fraction

The fractions represent a certain period of time between successive slot allocations. Due to this cyclical character, it is sufficient to know at which moment it starts, also called the start position, and its period in order to calculate future slot allocation according to this fraction. The start positions of the fractions is determined by means of a binary tree method, but can also be expressed as a formula. Formula (15) depicts the start position, Fpos_n, of a fraction f_n, expressed as the offset relative to the start position of the first fraction, f_0.

$$\text{Fpos}_n = \begin{cases} 0 & (n = 0) \\ \sum\limits_{i=0}^{n-1} \frac{1}{2}\frac{1}{f_i} & (n > 0) \end{cases} \tag{15}$$

with f_i being the unit fractions. Knowing that the unit of f_i is $\frac{1}{\text{frames}}$, the start position, Fpos_n, is expressed as the number of frames. The Formula (15) denotes that the offset is equal to half the sum of all periods of previous fractions. From this can be derived that the start position of fraction f_i occurs in the middle of the period of fraction f_{i-1}.

Appendix C

Proof of the maximum buffersize formula

We claim that the maximum buffer size can be calculated according to the following formula:

$$\text{Max\_buff}_k = R\frac{1}{f_0} + \sum_{i=1}^{k} \left(\frac{f_{i-1}}{f_i} - 2 \right) \frac{1}{f_{i-1}} \left(R - \sum_{j=0}^{i-1} f_j S \right) \tag{16}$$

with R being the requested bandwidth (bytes per frame), $f_0 \ldots f_n$ being the unit fractions that form the approximation, and S (bytes) representing the slot size.

This formula can also be written as:

$$\begin{aligned} \text{Max\_buff}_k = {} & \frac{1}{f_k} \left(R - \sum_{i=0}^{k-1} f_i S \right) \\ & + 2S - \frac{1}{f_0} R \\ & + 2S - \frac{1}{f_1} (R - f_0 S) \\ & \ldots \\ & + 2S - \frac{1}{f_{k-1}} \left(R - \sum_{i=0}^{k-2} f_i S \right) \end{aligned} \tag{17}$$

Proof:

We first prove this statement for a single fraction, followed by the proof for two fractions and finally the proof for k fractions.

Step 1: for a single fraction

A single fraction f_0 is sufficient to transmit the requested amount of data. This means that all data, collected at the requested rate during the interval between two subsequent transmissions, can be sent at once. Hence, the maximum buffer size is equal to the duration, $\frac{1}{f_0}$, multiplied by the rate, R:

$$\text{Max\_buff}_0 = \frac{1}{f_0} R$$

Step 2: for two fractions

Since there are two fractions necessary to approximate the requested bandwidth R, fraction f_0 is not sufficient to process all data. After each period, determined by $\frac{1}{f_0}$, the amount of accumulated data, which has arrived at a rate R, is larger than the slot size. Hence, not all data can be processed by the scheduled slots according to fraction f_0. The amount of remaining data after a period of $\frac{1}{f_0}$ is:

$$\text{Diff} = \frac{1}{f_0} R - S \tag{18}$$

During the period, specified by fraction f_1, $\frac{f_0}{f_1}$ slots are scheduled, which follow the allocation scheme of fraction f_0. The amount of data, which needs to processed, increases each time with *Diff* during $\frac{f_0}{f_1} - 1$ periods. The maximum buffer size is reached at the moment that an amount of data equal to *S*, is going to be sent in the slot that is scheduled according to f_0, before the slot that is scheduled according to f_1. The maximum buffer size is equal to:

$$\begin{aligned} \text{Max\_buff}_1 &= S + \left(\frac{f_0}{f_1} - 1\right) \text{Diff} \\ &= S + \left(\frac{f_0}{f_1} - 1\right)\left(\frac{1}{f_0}R - S\right) \\ &= S + \frac{f_0}{f_1}\frac{1}{f_0}R - \frac{f_0}{f_1}S - \frac{1}{f_0}R + S \\ &= \frac{1}{f_0}R + \left(2 - \frac{f_0}{f_1}\right)S + \frac{1}{f_0}\left(\frac{f_0}{f_1} - 2\right)R \\ &= \frac{1}{f_0}R + \left(\frac{f_0}{f_1} - 2\right)\frac{1}{f_0}(R - f_0 S) \end{aligned} \tag{19}$$

We have proven that for $f_0 + f_1$, Formula (16) holds.

One can also consider a different approach in order to prove Formula (16) for f_1. The methodology of using a series of fractions, uses the principle of splitting up the requested rate in a number of smaller rates. Each fraction is responsible for processing data at its own rate and thus lowering the remaining rate that needs to be processed. When starting from the highest fraction, the remaining rate needs to be lower than the processing rate of the fraction, otherwise, lower rates (fractions) will not be able to deal with this rate. During the period determined by the fraction f_i, data is arriving at a rate R_i, which is equal to the requested rate *R* minus the processing rates of the previous fractions f_0 ... f_{i-1}. During this period, an amount of data is accumulated, after which the fraction, f_i, is scheduled to lower the buffer size. The period of time that is spent for this process, can not be used for the accumulation of data. Hence, for fraction f_1, the maximum buffer size can be regarded as the amount of data arriving at a rate R_1, during a certain period $\frac{1}{f_1}$, minus the effort needed to lower the buffer size:

$$\text{Max\_buff}_1 = \frac{1}{f_1}R_1 - \text{effort}_1$$

Note that Formula (18) can be formulated as the amount of bandwidth per frame that yet needs to be scheduled by the second fraction. So the remaining bandwidth for fraction f_1, R_1, equals to:

$$R_1 = \frac{\left(\frac{1}{f_0}R - S\right)}{\frac{1}{f_0}} = \left(\frac{1}{f_0}R - S\right)f_0 \tag{20}$$

Knowledge about the moment where the maximum buffer size is reached is essential to understand the effort to lower the buffer size. The position of the first fraction, f_0, is considered as the reference position. The cyclic behavior of the slot allocation is defined by the lowest fraction. If the first slot that is allocated according to f_0 is at slot 0 of frame 0, we can say that the cycle starts at slot 0 of frames that are multiples of the lowest fraction. Due to the fact that the first slot for a certain fraction is allocated at the beginning of this cycle, the effort is located at the start of the cycle. It is the last fraction which ensures that all data is processed. This fraction has only a single slot allocated within the cycle, positioned at the beginning, since the cycle is equal to the period of the fraction. Note that the slots for every fraction f_i are positioned in the middle of the slots of the previous fraction f_{i-1}. Hence, the amount of data that arrived according to rate R_{i-1}, at the moment a slot is scheduled according to fraction f_i, is half the amount of data that arrives during a period $\frac{1}{f_{i-1}}$, that is:

$$\frac{1}{2}\frac{1}{f_{i-1}}R_{i-1}$$

From Formula (7), we know that this amount is smaller than the slot size S. Hence, the scheduling of fraction f_i results in a decrease of the buffer size. Since all these fractions start at the beginning of the cycle, the maximum buffer size is reached right before the start of the cycle. Since we have two halves, the next time a slot is scheduled according to f_{i-1}, another decrease of the buffer size takes place. Applying this to the current case, $f_0 + f_1$, the effort to diminish the buffer size is equal to:

$$\text{effort}_1 = 2\left(\frac{1}{2}\frac{1}{f_0}R_0 - S\right)$$

Therefore, the maximum buffer size is:

$$\begin{aligned} \text{Max\_buff}_1 &= \frac{1}{f_1}f_0\left(\frac{1}{f_0}R - S\right) - 2\left(\frac{1}{2}\frac{1}{f_0}R - S\right) \\ &= \frac{f_0}{f_1}\left(\frac{1}{f_0}R - S\right) - \frac{1}{f_0}R + 2S \\ &= \frac{1}{f_0}R + \frac{f_0}{f_1}\left(\frac{1}{f_0}R - S\right) - 2\frac{1}{f_0}R + 2S \\ &= \frac{1}{f_0}R + \left(\frac{f_0}{f_1} - 2\right)\left(\frac{1}{f_0}R - S\right) \end{aligned} \tag{21}$$

As can be seen, Formula (19) is equal to Formula (21).

Step 3: for k fractions

In order to calculate the maximum buffer size, we use the same approach as in the previous section. The maximum buffer size is the duration, determined by f_k, multiplied by the remaining bandwidth that needs to be scheduled, minus the effort to diminish the buffer size, i.e,:

$$\text{Max\_buff}_k = \frac{1}{f_k} R_k - \text{effort}_k$$

First, we will calculate the remaining bandwidth R_k. The remaining bandwidth R_1 was:

$$R_1 = f_0 \left(\frac{1}{f_0} R - S \right)$$

During the period $\frac{1}{f_1}$, a single slot is scheduled according to fraction f_1. Therefore, the remaining bandwidth per frame, R_2 equals to

$$\begin{aligned} R_2 &= f_1 \left(\frac{1}{f_1} \left(f_0 \left(\frac{1}{f_0} R - S \right) \right) - S \right) \\ &= f_1 \left(\frac{1}{f_1} (R - f_0 S) - S \right) \\ &= R - f_0 S - f_1 S \end{aligned}$$

For f_k, we can say that the remaining bandwidth per frame equals to:

$$R_k = R - \sum_{i=0}^{k-1} f_i S \qquad (22)$$

Using this formula, we can write the maximum buffer size for $k + 1$ fractions as:

$$\text{Max\_buff}_k = \frac{1}{f_k} \left(R - \sum_{i=0}^{k-1} f_i S \right) - \text{effort}_k$$

The effort$_k$ is a result of the scheduling of a slot, according to f_k, in the middle of two slots that are scheduled according to f_{k-1}, i.e. it is a result of the start positions of the fractions. From Formula (1), we know that the start position of fraction f_k is located at an offset of $\sum_{i=0}^{k-1} \frac{1}{2}\frac{1}{f_i}$, relative to the start of the first fraction, f_0. So, f_k starts in the middle of the period $\frac{1}{f_{k-1}}$, hence, the start position of f_k is equal to the start position of f_{k-1}, incremented by $\frac{1}{2}\frac{1}{f_{k-1}}$, that is, half the period according to fraction f_{k-1}. Therefore, we can say that the effort is equal to:

$$\begin{aligned} \text{effort}_k &= 2 \left(\frac{1}{2}\frac{1}{f_0} R_0 - S \right) \\ &+ 2 \left(\frac{1}{2}\frac{1}{f_1} R_1 - S \right) \\ &\ldots \\ &+ 2 \left(\frac{1}{2}\frac{1}{f_{k-2}} R_{k-2} - S \right) \\ &+ 2 \left(\frac{1}{2}\frac{1}{f_{k-1}} R_{k-1} - S \right) \end{aligned}$$

where R_0 ... R_{k-1} represents the remaining bandwidths for each of the fractions. By using Formula (22), we can write the effort as:

$$\begin{aligned} \text{effort}_k &= 2 \left(\frac{1}{2}\frac{1}{f_0} R - S \right) \\ &+ 2 \left(\frac{1}{2}\frac{1}{f_1} (R - f_0 S) - S \right) \\ &\ldots \\ &+ 2 \left(\frac{1}{2}\frac{1}{f_{k-2}} \left(R - \sum_{i=0}^{k-3} f_i S \right) - S \right) \\ &+ 2 \left(\frac{1}{2}\frac{1}{f_{k-1}} \left(R - \sum_{i=0}^{k-2} f_i S \right) - S \right) \end{aligned}$$

The resulting formula for calculating the maximum buffer size is:

$$\begin{aligned} \text{Max\_buff}_k &= \frac{1}{f_k} \left(R - \sum_{i=0}^{k-1} f_i S \right) \\ &- 2 \left(\frac{1}{2}\frac{1}{f_0} R - S \right) \\ &- 2 \left(\frac{1}{2}\frac{1}{f_1} (R - f_0 S) - S \right) \\ &\ldots \\ &- 2 \left(\frac{1}{2}\frac{1}{f_{k-2}} \left(R - \sum_{i=0}^{k-3} f_i S \right) - S \right) \\ &- 2 \left(\frac{1}{2}\frac{1}{f_{k-1}} \left(R - \sum_{i=0}^{k-2} f_i S \right) - S \right) \end{aligned}$$

Simplified, this result to:

$$\begin{aligned} \text{Max\_buff}_k &= \frac{1}{f_k} \left(R - \sum_{i=0}^{k-1} f_i S \right) \\ &+ 2S - \frac{1}{f_0} R \\ &+ 2S - \frac{1}{f_1} (R - f_0 S) \\ &\ldots \\ &+ 2S - \frac{1}{f_{k-1}} \left(R - \sum_{i=0}^{k-2} f_i S \right) \end{aligned} \qquad (23)$$

Q.E.D.

Abbreviations

CRC: Cyclic Redundancy Check; EDF: Earliest Deadline First; GPS: Generalized Processor Sharing; gcd: greatest common divider; lcm: least common multiple; MRBS: most regular binary sequence; MRCS: most regular code sequence; PGPS: packet-by-packet generalized processor sharing; TDMA: Time Division Multiple Access; WFQ: Weighted Fair Queuing; WSN: Wireless Sensor Network.

Competing interests
The authors declare that they have no competing interests.

Received: 13 March 2010 Accepted: 12 July 2011
Published: 12 July 2011

References

1. W Torfs, C Blondia, Binary TDMA Schedule by Means of Egyptian Fractions for Real-time WSNs on TMotes, in *Proceedings of Med-Hoc-Net 2010,* Juan-les-Pins, France (2010)
2. W Ye, J Heidemann, D Estrin, An Energy-Efficient MAC protocol for Wireless Sensor Networks, in *Proceedings of IEEE Computer and Communications Societies (INFOCOM)* (2002)
3. T van Dam, K Langendoen, An Adaptive Energy-Efficient MAC protocol for Wireless Sensor Networks, in *Proceedings of International Conference on Embedded networked sensor systems (SenSys), Los Angeles, California, USA* (2003)
4. L van Hoesel, P Havinga, Collision-free Time Slot Reuse in Multi-hop Wireless Sensor Networks, in *Proceedings of International Conference on Intelligent Sensor, Sensor Networks and Information Processing* (2005)
5. RA Rashid, W Embong, A Zaharim, N Fisal, Development of Energy Aware TDMA-Based MAC Protocol for Wireless Sensor Network System. Eur. J. Sci. Res **30**(4), 571–578 (2009)
6. V Turau, C Weyer, Scheduling Transmission of Bulk Data in Sensor Networks using a Dynamic TDMA Protocol, in *Proceedings of International Workshop on Data Intensive Sensor Networks (DISN)* (2007)
7. S Coleri-Ergen, P Varaiya, PEDAMACS: Power efficient and delay aware medium access protocol for sensor networks. IEEE Trans. Mob. Comput. **5**, 920–930 (2006)
8. http://en.wikipedia.org/wiki/Weighted_fair_queuing
9. http://www.sics.se/~ianm/WFQ/wfq_descrip/node21.html
10. A Demers, Analysis and Simulation of a Fair Queuing Algorithm, in *Internetworking Research and Experience* (1990)
11. A Parekh, R Gallager, A generalized processor sharing approach to flow control in integrated services networks: the single node case, in *IEEE INFOCOM* (1992)
12. http://en.wikipedia.org/wiki/Generalized_processor_sharing
13. L Dong, R Melhem, D Mosse, Time slot allocation for Real-Time messages with negotiable distance constraints, in *Real-Time Technology and Applications Symposium* (1998)
14. M Mock, E Nett, Real-Time Communication in Autonomous Robot Systems, in *Proceedings of International Conference on Autonomous Decentralized Systems* (1999)
15. JJ Babka, System and method for fractional resource scheduling for video teleconferencing resources. U.S. Patent 7,328,264 B2 (2008)
16. W Torfs, C Blondia, QoS support for a MAC with a TDMA tree topology on the Magnetic Induction Radio IC, in *Proceedings of Real-Time Systems Symposium (RTSS 2008), Barcelona, Spain* (2008)
17. http://en.wikipedia.org/wiki/Round-robin_scheduling
18. J Lessmann, GMAC: A position based energy efficient QoS TDMA MAC for Ad Hoc Networks, in *Proceedings of IEEE International Conference on Networks (ICON)* (2007)
19. CS Chen, WS Wong, Bandwidth allocation for wireless multimedia systems with most regular sequences. IEEE Trans. Wireless Commun **4**(2), 635–645 (2005)
20. K Gong, Egyptian Fractions, in *Math 196 Spring, UC Berkeley* (1992)
21. Algorithms for Egyptian Fractions http://www.ics.uci.edu/~eppstein/numth/egypt/intro.html

doi:10.1186/1687-1499-2011-37
Cite this article as: Torfs and Blondia : **Analysis of TDMA scheduling by means of Egyptian Fractions for real-time WSNs.** *EURASIP Journal on Wireless Communications and Networking* 2011 **2011**:37.