# Scalable and Adaptive Polling Protocol for Concurrent Wireless Sensor Data Flows

Manos Koutsoubelias, Antonis Argyriou and Spyros Lalis
*Electrical and Computer Engineering Department, University of Thessaly*
*Email: {emkouts, anargyr, lalis}@uth.gr*

***Abstract*—The uncoordinated transmission of sensor measurements to a collector over simple wireless technologies can lead to severely degraded performance when operating close to channel capacity. In this case, it can be far more effective to let the collector poll the sensor nodes to retrieve their data. We propose a coordinated protocol that exploits the broadcast capability of the wireless channel to eliminate contention between nodes and to minimize the number of packet transmissions performed at each poll. Furthermore, to let the protocol follow the dynamic changes in the sensor data generation rate, we propose an application-agnostic method for estimating the data generation rate of each node locally, and extend the protocol so that the collector dynamically adjusts the polling rate accordingly. Our method is based on a Kalman filter tuned for stable data generation rates, which is controlled via simple signals generated at runtime by the underlying polling protocol. We experimentally evaluate an implementation of our protocol for different data traffic scenarios using real nodes that communicate with the collector over IEEE 802.15.4 radio, showing that the collector can successfully track the actual data rate for both periodic and stochastic data generation.**

## 1. Inroduction

A wide range of monitoring applications, spanning from personal activity tracking to smart homes, employ wireless sensor nodes to obtain the required measurements. In most cases the sensors are sampled at a given rate, and the data, possibly after some local processing, is sent over the air to a collector. The collector, in turn, stores and processes this data to support some monitoring or decision/control process.

Several applications require this data transmission to be reliable. Also, in some occasions, data may need to be sent to the collector at a relatively high rate. There are two fundamentally different ways to achieve this. One approach is for each sensor node to send data to the collector as soon as this is produced. When the collector receives the data, it replies with an acknowledgement. We refer to this as *notification*. The other approach is for the collector to explicitly request the sensor nodes to send their data. In this case, each node stores the data in a local buffer, and transmits it only at the request of the collector. If the node receives a request but has no data, it replies with an empty acknowledgement. We refer to this method as *polling*.

Notification is attractive when there are a few sensor nodes and the data generation rate is relatively low compared to the capacity of the communication channel. Else, it can result in bad performance and poor utilization of the wireless channel. This is particularly so when nodes transmit in an uncoordinated way and the wireless communication technology does not feature any advanced medium access control mechanism.

In this case, polling can provide very good performance, especially when the data rate approaches the channel capacity, without requiring nodes to feature advanced radios or to be synchronized with each other. However, a key issue with polling is that the data generation rate may be unknown, and as a result the coordinator may poll too frequently or too slowly. Both situations are undesirable as they can waste system resources or reduce data freshness and lead to data buffer overflows, respectively.

To tackle the above issue, we propose an efficient polling protocol which dynamically adapts the polling rate of the collector according to the actual data generation rate of the sensor node. The main contributions of this paper are as follows: (i) We propose a polling protocol that exploits the broadcast capability of the wireless channel in order to reduce the number of packet transmissions required to retrieve data from sensor nodes as well as to minimize the respective latency. (ii) We propose an application-agnostic method for locally estimating the data generation rate of each node, based on a Kalman filter that is dynamically controlled/reset via signals that are generated at runtime by the polling protocol. (iii) We experimentally evaluate an implementation of our protocol in a IEEE 802.15.4 radio testbed showing that it can successfully track the actual application data rate.

The rest of the paper is structured as follows. Section 2 describes the basic polling protocol, and compares its peak performance to that of an equally simple notification protocol. Section 3 presents an application-agnostic data rate estimation method that can be applied locally by each node at the protocol layer of the wireless sensor nodes, while Section 4 discusses its experimental evaluation. Section 5 provides an overview of related work. Finally, Section 6 concludes the paper and points to directions for future work.

## 2. Basic Polling Protocol

Let processes $p_i, 1 \leq i \leq N$, represent a set of wireless sensor nodes that produce data. We refer to such processes as *sensors*. A special process $p_c$ represents the *collector*

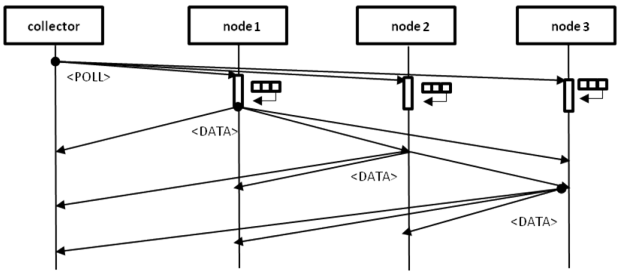


Figure 1: Basic polling protocol. When polled, nodes send data in sequence, in ascending order of their identifiers.

node. All processes are in the same broadcast domain, with a known maximum packet transmission delay $MsgT$.

When the collector wishes to retrieve data, it starts a poll by broadcasting a poll request. The set of sensors addressed is encoded in a bitmask $addr[1..N]$: if $addr[i] = 1$ then $p_i$ is expected to send a response, else $p_i$ should remain silent during the poll. The address mask also indicates the order in which sensors should respond. Namely, if both $p_i$ and $p_j$ are addressed in the poll and $i < j$, then $p_j$ must wait for $p_i$ to transmit, before it performs its own transmission.

Every sensor $p_i$ keeps the data it generates in a local buffer, and sends it when polled. If the buffer is empty, $p_i$ transmits an empty data packet. All data packets also include the size of the sender's data buffer. Just like the poll request, data packets are broadcast and are received/overheard by all sensors. Sensor $p_i$ responds to the poll as soon as it receives the packet of the immediately preceding sensor process $p_{prv}$ in the address bitmask. To deal with packet loss, $p_i$ sets a timer to expire when $p_{prv}$ is expected to have sent its data packet (based on the bitmask order and $MsgT$), and upon timeout proceeds with its own transmission (the timer is canceled if $p_i$ receives the packet sent by $p_{prv}$). If $p_i$ is first in order according to the address mitmask, it sends its data as soon as it receives the poll request.

Figure 1 illustrates the operation of a single poll. Each polling cycle, initiated by the collector periodically, may lead to several consecutive polls. These are performed at full speed, adjusting the address mask as needed, until the collector receives a reply from every sensor node and all nodes indicate that their data buffer is empty.

Note that the protocol serializes access to the shared wireless medium and eliminates contention, without relying on advanced medium access control mechanisms at the lower layers of the network stack. Thus, it is suitable even for simple radios and systems with a large number of sensors and high data rates. Also, sensors do not need to be synchronized or even have a proper clock.

To get a feeling about the performance of this protocol, Figure 2 shows the maximum sustainable aggregate and per-node reliable data transfer rate over IEE 802.15.4 radio, as a function of the number of sensor nodes that generate/send data. As a reference we use an optimal transport protocol where the sensor nodes send their data to the collector with prefect medium access synchronization. We also show the

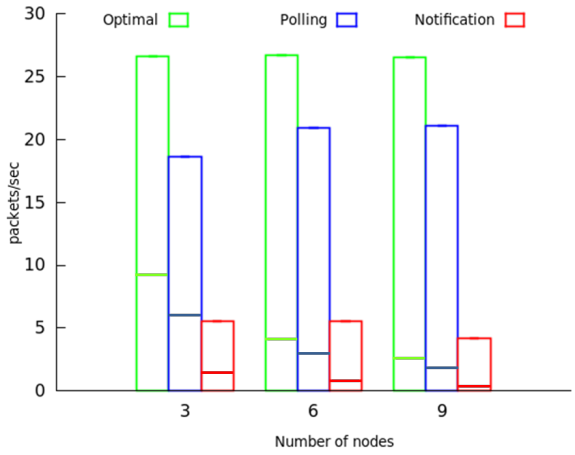


Figure 2: Maximum aggregate (full bar) and per-node (sub-bar) rate for reliable data transfer.

maximum data rate that can be sustained with acceptable data loss (less than $1\%$) for a notification protocol where each sensor node unicasts its data to the collector and waits for an acknowledgement.

It can be seen that the polling protocol approaches optimal performance as the number of nodes increases, because the overhead of the poll request is amortized over a larger number of data packet transmissions. In contrast, the performance of the notification protocol is bad and degrades to the number of sensor nodes, due to the increasing contention.

## 3. Rate Estimation

If the collector knows the rates at which sensor processes (nodes) generate data, it can set its polling strategy accordingly. However, in the general case, the data rate $r_i$ of sensor process $p_i$ may be unknown, even to the process itself. For instance, the application layer may not inform the protocol layer about the rate at which data will be generated, because it may be not be able or willing to compute it. The challenge is for the protocol layer to estimate $r_i$ without any input from the application.

We assume that data generation is independent for each sensor process, and let each $p_i$ estimate $r_i$ locally. The estimated rate is then piggy-backed to the response that $p_i$ sends to the collector in order to adjust the polling rate.

In the general case, the average data rate $r_i$ may vary in time. We assume that changes in the average data rate are rather abrupt, corresponding to internal state changes of the application (which are hidden from the protocol layer). We also assume that once such a state change takes place, the application remains in that state for a relatively long time. These assumptions are realistic for a wide range of applications. Two indicative scenarios are illustrated in Figure 3: the top part shows a process that performs a transition between different but constant data rates, while the bottom part illustrates a transition between two different average rates for a stochastic data generation process.

Our goal is to design a unified method that can estimate the average rate in both cases, without knowing which case actually applies to the current system operation. To describe our approach, we introduce additional notations that capture the time-dependency of the average data rate. We denote with $r_i[n]$ the actual average data rate of the application process when the $n$-th data packet is generated. Process $p_i$ can estimate $r_i[n]$ in different ways depending on the assumptions about the model that governs data arrivals.

For our purposes, we assume a first-order dynamic model for the the evolution of the average arrival rate:

$$r_i[n] = a \times r_i[n-1] + u[n],\ n \geq 0 \quad (1)$$

The value of parameter $a$ corresponds to the weight of the last average rate of the application: if $a$ is close to 0 then the previous value hardly affects the next one, whereas if $a$ is close to 1 then the previous value strongly affects the next value. In this model $u[n]$ is a random variable that follows a Gaussian distribution, and models potentially abrupt changes in the average rate over successive time instants. Since our assumption is that most of the time the application generates data at a constant average rate, the best choice is to set $a$ close to 1.

We want to compute an estimate of $r_i[n]$. However, this estimate will unavoidably suffer from measurement noise: $x_i[k] = r_i[k] + w[k]$. The question is how to estimate $r_i[n]$ accurately. The metric we use is the mean square error (MSE) from the Bayesian estimation theory.

Since $u[k]$ is Gaussian for the linear model in (1), we adopt the scalar Kalman filter, which has been shown to be optimal for the linear data model [1]. The scalar Kalman algorithm is an adaptive filter defined from a set of recursive equations:

$$\text{Prediction}: \hat{r}[n|n-1] = a\hat{r}[n-1|n-1]$$
$$\text{Min. Pred. MSE}: c_e[n|n-1] = a^2 c_e[n-1|n-1] + \sigma_u^2$$
$$\text{Kalman Gain}: K[n] = \frac{c_e[n|n-1]}{c_e[n|n-1] + \sigma_u^2}$$
$$\text{Correction}: \hat{r}[k] = \hat{r}[n|n-1] + K[n](x[n] - \hat{r}[n|n-1])$$
$$\text{Minimum MSE}: c_e[n|n] = (1 - K[n])c_e[n|n-1]$$

The algorithm initializes $c_e[n-1|n-1]$ to an arbitrarily low value close to 0. Then it calculates recursively $K[n], \hat{r}[n], c_e[n|n]$.

The above model will work well, regardless of whether data generation is periodic or stochastic (Poisson), provided that the average data rate remains constant, and $a$ is close to 1 (which is true in our case). However, as already noted, we are interested in supporting scenarios where abrupt changes in the average data rate can occur. One way to do this is to employ a more "clever" filter that can detect and respond to such transitions. Here, we follow a different approach, namely we keep the same simple filter, but *control* its operation based on information that is available at the layer of the polling protocol. The heuristic is quite intuitive: since the above filter is not designed to track significant changes in the average data rate, it is reset when there is an indication of inefficient polling. More concretely, each node resets its Kalman filter when the collector performs several consecutive poll cycles that do not return any data (collector polls too fast), or when in a given poll cycle the collector performs several consecutive polls in order to empty the buffers of the data processes (collector polls too slowly). As it will be shown in the next section, it suffices to adapt the operation of the filter, and to let the collector adjust the polling rate according to the data rate estimates received from the sensor nodes.

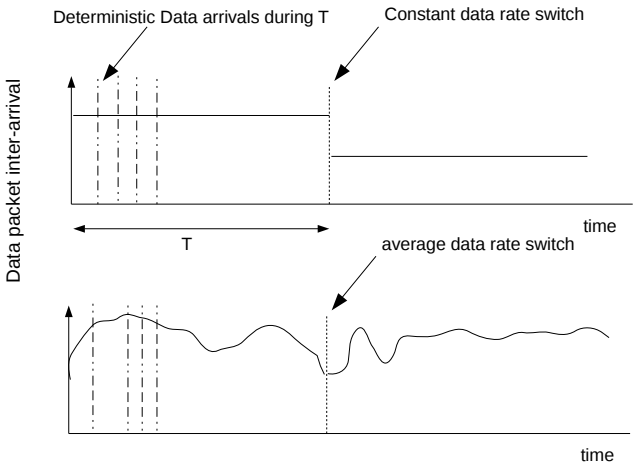


Figure 3: Traffic models considered in this work.

## 4. Experimental Evaluation

The proposed polling protocol has been implemented as a refinement of GCBRR [2], which supports 1-to-N request-reply interactions in a process group. We have evaluated our implementation using real wireless nodes that communicate over 802.15.4 radios. In the following, we describe the test setup, and then present results from indicative experiments.

### 4.1. Experimental setup and metrics

For our experiments we use TelosB-class devices from AdvanticSys [3], equipped with 802.15.4 radios. To enable detailed logging during the experiments, the protocol logic runs on powerful PCs, which are connected over serial to the TelosB devices that essentially act as wireless modems. We have confirmed that the serial interface is fast enough to support the packet rate of 802.15.4. Henceforth we refer to a PC-TelosB pair as a "node".

The nodes are arranged in a 1-hop topology so that they can communicate with each other directly. A distinguished node is the collector, all others play the role of sensor nodes that produce data. Sensor nodes run a simple application that generates data items according to a predefined scenario. The scenario is stored in a file as a sequence of time intervals. When the experiment starts, the application waits for a short random back-off interval and then, in a loop, reads the next

entry from the scenario file, sleeps for the specified amount of time, and issues the next data item for transmission to the collector. We investigate both periodic/deterministic and stochastic data generation scenarios. In the latter case, the scenario file is generated using Matlab, according to the behavior of a Poisson process (separate scenario files are genenerated for each sensor node). The data produced by the application are passed to the protocol layer for transmission. In our experiments we let each data item occupy a full packet. As a consequence, it is not possible to piggy-back several data items in a single packet, and if a node has more than one items buffered locally then the collector has to retrieve them via consecutive polls that are performed within the same polling cycle.

The protocol layer of the node runs the heuristic for estimating the local application data rate, as discussed in Section 3. This information is included in the data packets sent to the collector when the node is polled. Based on this information, the collector adjusts the rate at which it performs polling cycles, also referred to as *polling rate*. The strategy of the collector is intentionally kept simple, namely the polling rate is set equal to the highest data rate reported by a sensor node. Note that this is suboptimal from a latency perspective, because a data item will remain in the node's buffer for half the polling period on average, and for an entire polling period in the worst case. Since the objective of our experiments is to evaluate the ability of the protocol to adapt its polling cycle according to the average application data rate, we do not try to optimize the delivery delay.

## 4.2. Unknown but stable data rates

In a first set of experiments, we evaluate the ability of the polling protocol to track a stable but unknown data rate of the sensor nodes. We run the rate estimation heuristic with the filter reset logic disabled (no-reset version). We have experimented with different application data rates and numbers of sensor nodes, for periodic as well as stochastic (Poisson) data generation scenarios. Due to space constraints, here we present the results for a rate of 0.5 data packets/second and 3 nodes (other configurations result in similar behavior). The results are shown in Figure 4 and Figure 5. The top plot in these figures shows the actual application data rate of each node and the rate at which the collector decides to perform polling cycles (polling rate), averaged over 10-second intervals. The bottom plot shows the total number of polls and the number of void polls performed by the collector over 10-second intervals (recall that each polling cycle may include several consecutive polls).

In the periodic data generation experiment, after a short warm-up phase, the sensor nodes accurately estimate their local application rate. As a consequence, the collector performs polling cycles at exactly the right rate, avoiding any void polls. The same observation regarding the accuracy of the estimated data rate also applies to the stochastic data generation scenario. However, in this case, due to the inherent variance of the application data generation process, some sensor nodes end-up having several data items in

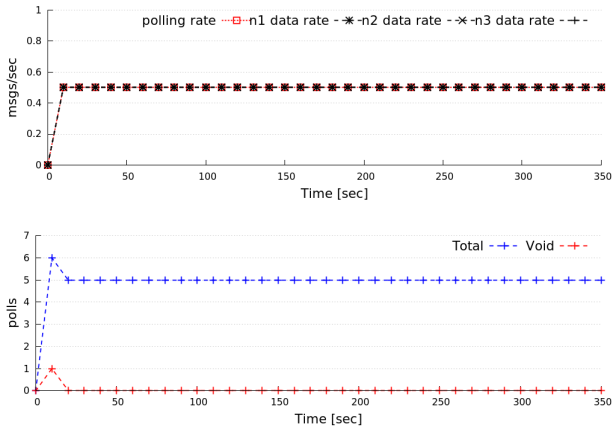


Figure 4: Periodic data generation.

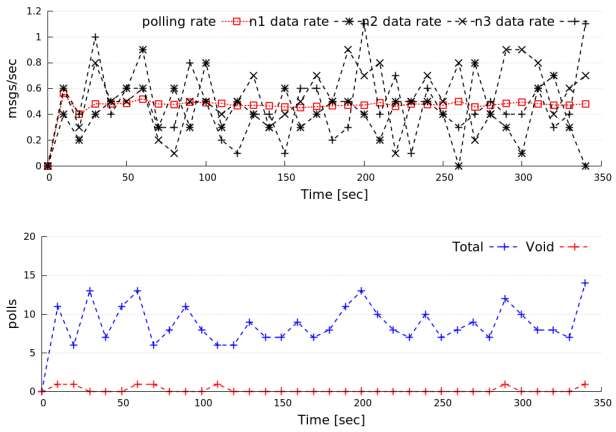


Figure 5: Stochastic data generation.

their buffer, and the collector often has to perform several polls per cycle to retrieve this data. The variance in data generation also results in the opposite effect, namely it can be that no node generates any data within the next polling interval, in which case the collector will perform a void poll. But, naturally, this happens quite rarely.

## 4.3. Unknown and dynamically changing data rates

In a second set of experiments, we evaluate the ability of the polling protocol to track dynamic transitions between significantly different unknown data rates of the application. Here, we test the rate estimation heuristic with the filter reset logic disabled and enabled (no-reset vs. reset version). We investigate a scenario where the sensor nodes initially have a data rate of 0.5 data packets/second, then switch to a data rate of 1.5 data packets/second, and after some time switch back to the initial data rate. Due to space limitations we only present the results for 3 nodes, for periodic as well as for stochastic data generation.

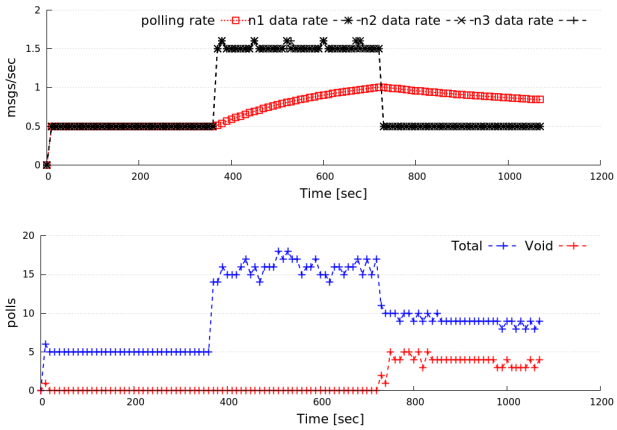


Figure 6: Periodic data generation with dynamic rate transitions, filter reset disabled.

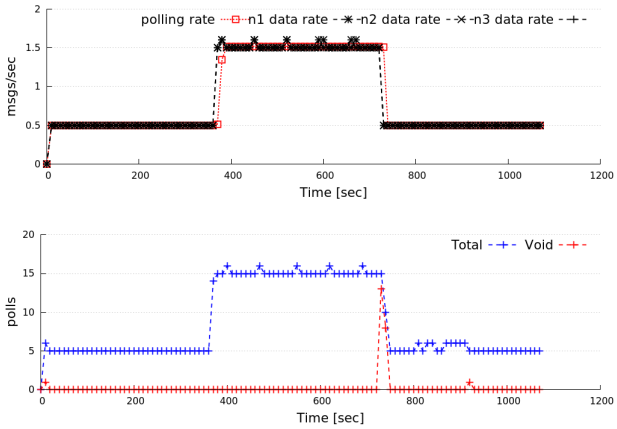


Figure 7: Periodic data generation with dynamic rate transitions, filter reset enabled.

The results for the periodic data generation scenario are shown in Figure 6 and Figure 7. The no-reset version performs well in the first phase of the experiment, where the data rate does not change. However, when the application switches to a different rate, the estimation filter reacts slowly and is unable to find the actual data rate. The inability to track the first rate switch results in infrequent polling cycles, which leads to a slightly increased number of polls compared to the node's data rate. Since nodes generate data at different points in time, their buffers increase unevenly in time, and the number of polls in a polling cycle is determined each time by the node with the longest buffer. Conversely, after the second switch, inaccurate data rate estimation results in overly frequent polling, and to a significant number of void polls.

In contrast, the reset version tracks the rate transitions of the application smoothly and accurately. Thus the collector always polls the sensor nodes at the right rate. There are just a few void polls when the application makes the transition

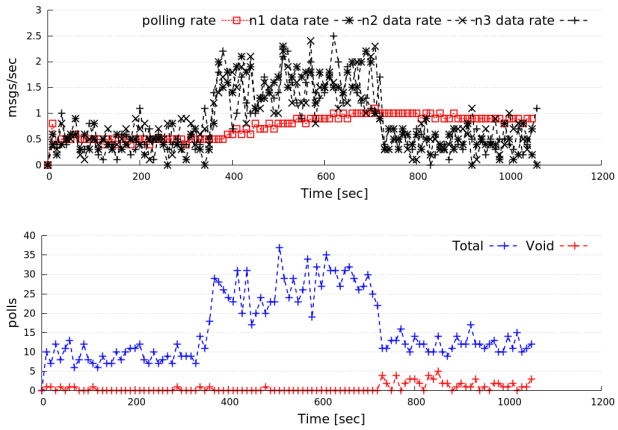


Figure 8: Stochastic data generation with dynamic rate transitions, filter reset disabled.

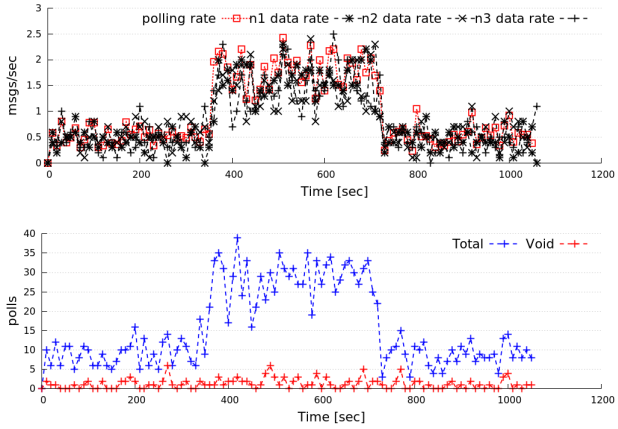


Figure 9: Stochastic data generation with dynamic rate transitions, filter reset enabled.

from the higher to the lower data rate, at which point the filter is reset and can immediately track the new data rate.

The results for stochastic data generation are shown in Figure 8 and Figure 9. Again, the no-reset version fails to track the rate changes of the application. The reset version is (as before) better in this respect, but now this does not always translate in better behaviour. Namely, due to the variance of data generation, the reset version adjusts the polling rate in a jerky way, even when the average rate is stable, and performs more void polls than the no-reset version in the first and second phase of the experiment. In the third phase, when the application switches from a high to a low rate, the more accurate estimation of the application data rate pays-off, leading to a reduced number of total polls and void polls compared to the no-reset version.

## 5. Related Work

The efficient data transfer in WSNs where the channel capacity is limited has been studied at various layers of the network stack. S-MAC [4] uses CSMA to eliminate packet collisions, and synchronizes the transmiting nodes using a sleep-wakeup schedule. T-MAC [5] follows a similar approach, but it employs an adaptive logic for the sleep-wakeup schedules based on a timer which is adjusted based on the network load. Both approaches are designed to be energy efficient, however they do not perform well for high network loads.

Z-MAC [6] is adaptive to the network load. It uses CSMA for low network loads, and switches dynamically to a version of TDMA when high network load is sensed. In the TDMA mode, nodes can use empty slots based on a priority scheme where the owners of the slots have higher priority than others. Z-MAC achieves high throughput, but it requires clock synchronization among the senders, and needs extra carrier sensing for the utilization of empty slots.

Another protocol that tolerates variations in periodic network load of a sensor network is SCP-MAC [7]. In this case, the nodes are sleeping and periodically wakeup to perform channel polling. When a node has data to transmit, it first sends a short preamble to the respective destination, which in turn is activated to receive the data. The nodes are strictly synchronized in a fixed schedule in order to wakeup for the preambles. Prior to the transmission of the preamble, the medium is sensed to avoid collisions between multiple senders. A more adaptive approach is adopted in WiseMAC [8], where each node learns the sampling period of its neighbours, and creates a local wakeup schedule that is piggy-backed to each data transmission. The receiver combines its own schedule to the one received, to minimize void wakeups (getting ready for packet reception even though no node will transmit).

In terms of reliability, a number of protocols have been proposed above the MAC layer. In RCRT [9] several reliable unicast flows are directed from the nodes to a sink, which is responsible for controlling their data rate. The protocol guarantees reliability as the missing packets are recovered through a negative acknowledgement scheme. However, as the network size increases, the supported transmission rates drop significantly due to high contention. RBC [10] also uses negative acknowledgements, but takes a different approach to decrease channel contention by giving priority to the nodes that have more data to transmit. Nodes piggyback their buffer size to each packet, while other nodes overhear these transmissions and postpone their own data transmission for a period of time if the received buffer size exceeds a certain threshold. This technique avoids contention, but nodes with a higher data rate might lead to the starvation of nodes with lower rates.

Our approach does not rely on a specific wireless technology, can be applied on top of simple radios, does not require clocks or any elaborate synchronization between the transmitting nodes, supports high data rates close to channel capacity with high reliability without explicit acknowledgements while performing very few (usually no) retransmissions, is fair for all nodes, and can be extended to support priorities without starvation. Also, thanks to the ability of the protocol to dynamically adapt the polling rate, the number of unnecessary polls and empty replies sent by sensor nodes can be reduced when the application switches to low data rates.

## 6. Conclusions

We have presented a simple but effective polling protocol, designed to allow even a large number of sensor nodes to send data over the air to a collector, without contention and good utilization of the wireless channel. We have also proposed and evaluated an application-agnostic mechanism for estimating the actual data rate so that the collector can adjust the polling rate accordingly.

The protocol can be extended to allow sensor nodes to send several packets as a response to a single poll request. Especially in stochastic data generation scenarios, this can greatly reduce the number of consecutive polls that need to be performed in order to empty the data buffers of the sensor nodes, further amortizing the cost of the poll request. It is also possible to devise more elaborate strategies for the collector, to trade-off polling efficiency (and the number of void polls) for improved data freshness.

## References


[1] S. M. Kay, *Fundamentals of Statistical Signal Processing, Volume I: Estimation Theory*. Prentice Hall, 1993.

[2] M. Koutsoubelias and S. Lalis, "Coordinated Broadcast-based Request-Reply and Group Management for Tightly-Coupled Wireless Systems," in *Proceedings of the 22nd International Conference on Parallel and Distributed Systems*, 2016, pp. 1163–1168.

[3] [Online]. Available: https://www.advanticsys.com/shop/mtmcm5000sma-p-23.html

[4] W. Ye, J. Heidemann, and D. Estrin, "An energy-efficient mac protocol for wireless sensor networks," in *Proceedings of the 21st Annual Joint Conference of the IEEE Computer and Communications Societies*, vol. 3, 2002, pp. 1567–1576.

[5] T. van Dam and K. Langendoen, "An adaptive energy-efficient mac protocol for wireless sensor networks," in *Proceedings of the 1st International Conference on Embedded Networked Sensor Systems*, 2003, pp. 171–180.

[6] I. Rhee, A. Warrier, M. Aia, J. Min, and M. L. Sichitiu, "Z-mac: A hybrid mac for wireless sensor networks," *IEEE/ACM Transactions on Networking*, vol. 16, pp. 511–524, 2008.

[7] W. Ye, F. Silva, and J. Heidemann, "Ultra-low duty cycle mac with scheduled channel polling," in *Proceedings of the 4th International Conference on Embedded Networked Sensor Systems*, 2006, pp. 321–334.

[8] A. El-Hoiydi and J.-D. Decotignie, "Low power downlink mac protocols for infrastructure wireless sensor networks," *Mob. Netw. Appl.*, vol. 10, no. 5, pp. 675–690, Oct. 2005.

[9] J. Paek and R. Govindan, "Rcrt: Rate-controlled reliable transport protocol for wireless sensor networks," *ACM Trans. Sen. Netw.*, vol. 7, no. 3, pp. 1–45, 2010.

[10] H. Zhang, A. Arora, Y.-r. Choi, and M. G. Gouda, "Reliable bursty convergecast in wireless sensor networks," in *Proceedings of the 6th ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2005, pp. 266–276.