MMEDD: Multithreading Model for an Efficient Data Delivery in wireless sensor networks

Blaise Omer Yenke[1], Damien Wohwe Sambo[2], Ado Adamou Abba Ari[3] and Abdelhak Gueroui[4]

[1 2]LASE Laboratory, University of Ngaoundéré, Cameroon
[3]Mathematics and Computer Science Department, FS, University of Maroua, Cameroon
[3 4]LI-PaRAD Laboratory, Université Paris Saclay, University of Versailles Saint-Quentin-en-Yvelines, France

Abstract**:** Nowadays, the use of Wireless Sensor Networks (WSNs) is increasingly growing as they allow a large number of applications. In a large-scale sensor network, data transmission among sensors is achieved by using a multihop communication model. However, since its resources limit the sensor, sensors' Operating Systems (OSs**)** are developed in order to optimize the management of these means, especially the power consumption. Therefore, the hybrid operating system Contiki uses a low consumption layer called Rime, which allows sensors to perform multihop sending with a low energy cost. This is favored by the implementation of lightweight processes called protothreads. These processes have a good efficiency/consumption ratio for monolithic tasks, but the management of several tasks remains a problem. In order to enable multitasking, Contiki provides to users a preemptive multithreading module that allows the management of multiple threads. However, it usually causes greater energy wastage. To improve multithreading in sensor networks, a Multithreading Model for an Efficient Data Delivery (MMEDD) using protothreads is proposed in this paper. Intensive experiments have been conducted on COOJA simulator that is integrated in Contiki. The results show that MMEDD provides better ratio *message reception rate/energy consumption* than other architectures.

Keywords**:** Multithreading, Multitasking, Protothreads, WSNs, COOJA, MMEDD**.**

1. Introduction

The technological developments carried out in wireless networking have created a new generation of networks constituted of small entities called sensors, which are capable of gathering information from the environment in spite of their limited computing, memory and storage resources [1]. In addition, progress in microelectronics and wireless communications allowed the production of sensors in reasonable costs [2, 3]. As the sensors have increasingly small sizes, their communication range remains limited. In order to cover a large area, sensors are deployed and connected to each other, thereby forming a Wireless Sensor Network (WSN). A WSN usually consists of a deployment of one or more static or mobile sink nodes and a number of sensor nodes on a physical environment [4, 5]. In such a network, each node is able to gather, process physical information and transmit the gathered data to a remote Base Station (BS) through a sink-node [6].

However, sensors are designed with resource constraints such as a restricted computing capacity; reduced memory size and storage; weak range of communication; low bandwidth and the limited amount of energy [7]. The energy resource is the most important parameter to be considered in the design of a WSN since it first defines the sensor's lifetime and then the whole network lifetime [1, 8].

Behind the energy constraint, the limited memory size and storage constraints overstrain sensors to execute light and not complex programs [9]. The OS embedded in the sensor realizes the management of these constraints. Two main types of OS exist in Wireless Sensor Networks (WSNs): Multitask (thread-based) and Event (event-based) systems [10]. The main problem of Multitask systems is the allocation of memory to different processes, but they permit the simultaneous execution of tasks with a non-negligible energy wastage. Nevertheless, event-based systems enable a better memory management, thus observe low power consumption, but do not allow long treatment and complex tasks [11].

In order to reduce the disadvantages of these systems, hybrid systems such as Contiki OS have been built [12]. The hybrid conception of Contiki allows to observe low energy consumption like event-based system due to its lightweight processes protothreads, which are designated for the multitask treatment. Contiki supplies to developers an optional multithreading library. However, it requires more memory resource and hence will consume a lot of energy.

In this paper, in order to reduce the energy consumption during a multithreading treatment, a multithreading model for an efficient data delivery in WSNs called MMEDD is proposed. To achieve the proposal, the lightweight protothread included in Contiki is used. The performance analysis shows that the proposed architecture has approximately the same data delivery rate than the threaded model. Moreover, the proposal enables less energy consumption than the other models. In short, our main tasks can be summarized as follows:

- Evaluation of the energy consumed during the data delivery, using the native multithreading library;
- Comparison of the obtained results with those without multithreading;
- Modelling an architecture for the multithreading by using protothreads instead of threads;
- Simulation of the MMEDD in order to highlight its performance.

The rest of this paper is organized as follows: Section 2 presents sensors' architectures and OS; Section 3 formulates the problem of using multithreading in WSN, and provides the performance analysis of data delivery and the energy consumption; in Section 4, the design goals and the description of the MMEDD are given; an experimental validation of our architecture is provided in Section 5; conclusion and directions for future work are presented in Section 6.

2. Sensors' Architectures and OS

In this section, we first discuss about WSNs' architectures especially the different types of WSNs. Then, some existing OS dedicated to sensors are reviewed.

2.1 WSNs' architectures

A WSN can be defined as the combination of two main types of sensor nodes within the same network: the simple node and the gateway node [1, 7]. The simple nodes gather information from the sensing field and transfer the gathered data to the gateway nodes, which are linked to a remote BS via Internet or a LAN.

Communication within the sensors' field is done in a single or multi hop manner. Communication is done in multihop when two distant nodes communicate through an intermediate node while communication is done in a single-hop when there is no intermediate node [3]. The architecture of WSN is directly linked to the configuration of the sensor field. Ari et al. [1] present two main architectures in WSNs: flat architectures and hierarchical architectures.

In flat architecture, apart from the sink node, other nodes are homogeneous. Nodes communicate with the sink either in single or in multihop manner. Moreover, large-scale sensor networks usually use hierarchical architectures. In these architectures, the network is subdivided into several groups of sensors, usually sub-networks called clusters. A special node called Cluster Head (CH) represents each cluster. Interesting studies have been proposed for cluster formation and CH elections in WSNs [13, 14]. CH has to aggregate and/or compress the collected data and transfer the aggregated data to sink [15, 16], according to secured routing protocols such as the work proposed in [17].

2.2 Sensors' OS

In a WSN, an OS is defined as a light layer of software that is located between the hardware and the application, which enables basic programming abstractions for developing applications [2]. The main aim of OS for WSNs is to allow applications to communicate with material resources, to schedule and prioritize tasks, and to ensure the regulation between conflict of applications and services. Operating systems' functionalities include: power and memory management; file management; networking communication; and programming environments that allow building applications.

Traditionally, OS are classified as single-task or as multitasking OS. Single-task OS executes one task at a specific time whereas multitasking OS can process many tasks simultaneously. The multitasking OS allows a sensor node to receive data from the sensing unit and to deliver data in parallel. However, a multitasking OS requires a large amount of memory, but sensor nodes are limited by its resources [2]. TinyOS [18] and Contiki [12] are the most used OS specifically designed for sensors [19]. TinyOS is an event-driven OS, its middleware supports time synchronization, routing, data aggregation, localization, radio communication, task scheduling, I/O processing, etc. The multitask is implemented in TinyOS in the TOSThread module. TinyOS is designed to run on equipment's that have very low memory capacity [18]. Moreover, Contiki is a hybrid-driven OS combining event and thread functions. Contiki observes a small memory footprint as TinyOS due to the use of a lightweight process called protothread. The multitasking management is made possible through a multithreading module and its integration is optional because it uses more memory resources.

Apart of TinyOS and Contiki systems, other OS for WSNs exist, based most of the time on a multi-threaded semantic model. However, these systems do not allow low energy consumption because they use locking mechanisms to achieve mutual exclusion of shared variables, while TinyOS and Contiki use an event-based scheduler without preemption. In the literature the following systems can be found: Nano-Qplus [10], PAVENET OS [10], SOS [20], RETOS [21], LiteOS [22], Nano-RK [23], etc. Moreover, Liu et al. [24] have done a multithreaded comparison of the RAM usage between the systems TinyOS and Contiki. They show that multithreading in Contiki is lighter than the TOSThread of TinyOS. In the rest of the document we focus on Contiki OS because of its protothreads properties and the enabled multithreading performance.

3. Multithreading in WSNs

This section describes in detail the problem of using the multithreading in a WSN. An energy consumption analysis between a classical architecture and that using the multithreading module is also carried out.

3.1 Problem statement

The problem of power and resource management is overriding for WSNs. In traditional multithreaded OS, the size of each stack is ingeniously reserved in order to avoid memory overflow and memory wastage. The main problem of multithreaded OS is that every created thread runs in a pre-reserved stack heuristically. Then if the reserved stack size is too small for a running application, it will generate a memory overflow. To solve this problem, the stack size needs to be assigned a large value that can meet the requirement of the worst case [24]. However, when the thread does not require a large stack size, the memory space reserved will not be efficiently used, leading to high energy wastage as shown on Figure 1.

3.2 Performance analysis

This Section presents the experiments carried out on the Rime layer of Contiki OS by analysing the consequences of using threads for data delivery in WSN.

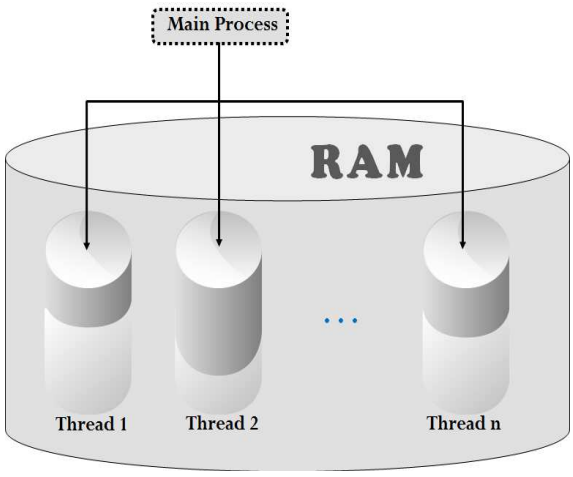

Figure 1. Thread implementation

3.2.1 *Adopted methodology*

In order to perform the analysis of the implemented thread module for data delivery in Contiki, we consider an example of a multihop communication by using the default library *multihop.h* of the Rime layer. To conduct the analysis, we consider a central node (inter) that will receive packets sent by some nodes (sender) and transfer these packets to another receiver-node (receiver). The idea is that a sender-node should not communicate directly with a receiver-node due to their limited communication range like highlighted in Figure 2. Thus, the sender-node transmits a packet in multihop through inter-node. Simulation was realized with the simulator COOJA integrated to Contiki-2.7. Sky-mote nodes have been used and the coordinates given in Table 1 provide the random position.

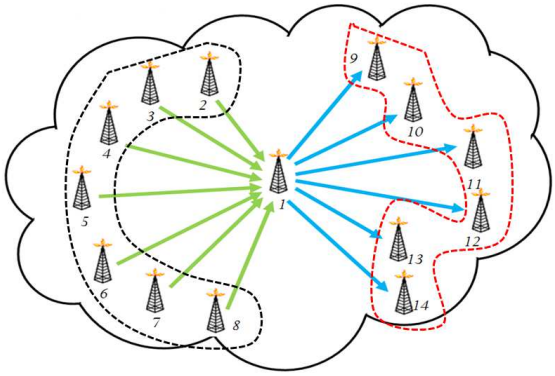

Figure 2. Experimental network

Firstly, we have used the classical architecture in which the retransmission of packets by the inter-node sequentially follows the FIFO scheduling mechanism as shown on Figure 3**.** Secondly, we have modified the library *multihop.h* so that, after receiving a packet to be transferred, the inter-node, creates a thread responsible for delivering packets to the receiver-node. The duty cycling implemented is defined by the ContikiMAC which is set as default by COOJA. The multi hop thread-based data transfer is presented in Figure 4.

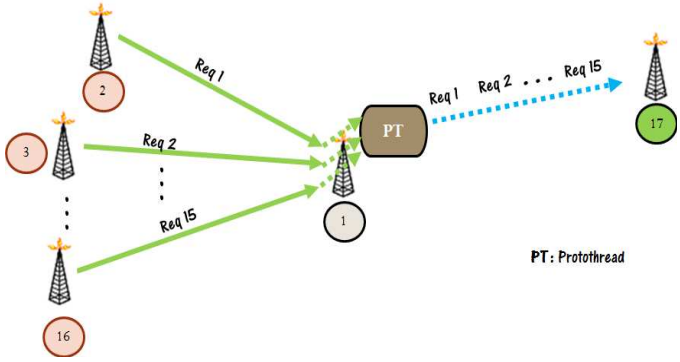

Figure 3. Multihop transfer using the Classical architecture

For practical purposes, the average time between two cycles of the sender-node is fixed at *2ms*. A cycle corresponds to the transmission of a sender. We have varied the number of senders between 1 and 15. The number of senders is set to 15 in order to avoid sensors redundancy on the considered sensor field. We conducted more than 10 tests over 30 cycles. The results were quite the same. Then, we analysed the number of received messages and the energy consumption of inter-node using the classical and the thread-based architecture.

Table 1. Nodes' Coordinates

Node *id*	Axis (x-x')	Axis (y-y')
1	95.099	55.277
2	63.892	33.496
3	63.413	50.411
4	105.958	86.986
5	68.541	80.854
6	77.718	91.932
7	76.697	62.781
8	88.096	71.909
9	70.237	54.939
10	95.685	92.837
11	95.403	81.934
12	68.188	45.108
13	101.094	81.764
14	70.049	36.727
15	73.013	50.47
16	82.314	68.921
17	123.692	32.81

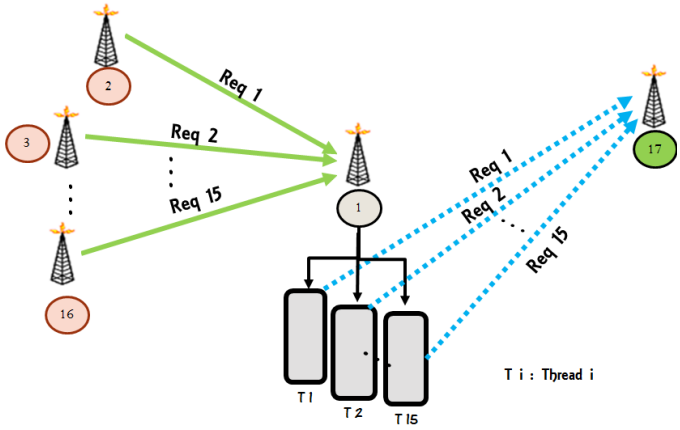

Figure 4. Thread implementation

3.2.2 *Reception message rate*

Results on Table 2 present the number of messages received by the inter-node over 30 cycles between classical and the thread-based architectures. From the data on Table 2, the reception rate for thread-based and classical architecture was computed. We obtained 72.33% for the thread-based architecture against 63.74% for the classical architecture. This shows that the data delivery using thread-based architecture achieves 8.59% messages more than classical architecture. These results are clearly plotted in Figure 5. This difference of performance is explained by the fact that the thread-based architecture receives and retransmits faster than the classical architecture.

Table 2. Amount of received message on 30 cycles – Classical Vs Thread-based

Senders	Msg-Send	Classical	Threads
1	30	30	30
2	60	42	56
3	90	57	70
4	120	90	94
5	150	90	120
6	180	134	151
7	210	146	165
8	240	177	179
9	270	179	202
10	300	172	205
11	330	201	213
12	360	198	218
13	390	189	177
14	420	169	237
15	450	188	218

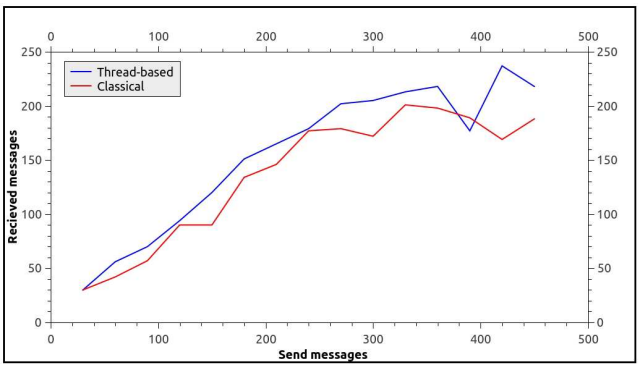

Figure 5. Comparison of the amount of received messages on 30 cycles – Classical Vs Thread-based

3.2.3 Energy model

We adopted the energy model obtained by Sehgal [25] (Equation 1), which allow estimating the energy consumption of the inter-node.

$$Power(mW) = \frac{(rx + tx + ON)}{\gamma * runtime} * i * v \qquad (1)$$

Where:

- *rx*, *tx* and *ON* respectively represent the time passing by the radio in the receive mode, transmit mode and the Cpu_ON;
- $\gamma = 4096$ represents the number of ticks per second;
- *runtime* is evaluated in seconds;
- $i = 20\ mA$ is a pre-measured value available on the data sheet;
- $v = 3V$ is the approximated value of Sky-mote operational voltage.

Table 3 shows the results in milliWatts (*mW*) of the energy consumption on the inter-node in the classical as well as in the thread-based architectures. The average energy consumption between these two architectures is slightly the same (*7.3780 mW* for classical architecture against *7.3074 mW* for the thread-based architecture).

Table 3. Energy consumption on 30 cycles – Classical Vs Thread-based

Senders	Classical	Threads
1	2.5956	4.5877
2	3.1021	3.4757
3	3.0470	5.7267
4	3.9621	4.70905
5	4.5368	5.1678
6	7.2492	6.4264
7	9.3301	9.6382
8	7.9583	10.2538
9	6.1979	8.1779
10	8.4618	9.1186
11	10.3861	10.1320
12	12.9091	9.3255
13	12.8760	8.0336
14	8.6349	5.5132
15	9.4236	9.3250

The overlap observed in Figure 6 does not show a greater difference between the classical and the thread-based architecture. We observe that on average, the classical architecture consume less energy than the thread-based architecture, when the number of senders is less than 10. However, when the number of senders increases (more than 10), the classical model has higher energy consumption than the thread-based. This can be explained by the fact that, on classical architecture, the received messages have to be forwarded to receiver in FIFO order and then queued. The limited size of RAM on WSN influences the number of messages that can be stored on the queue.

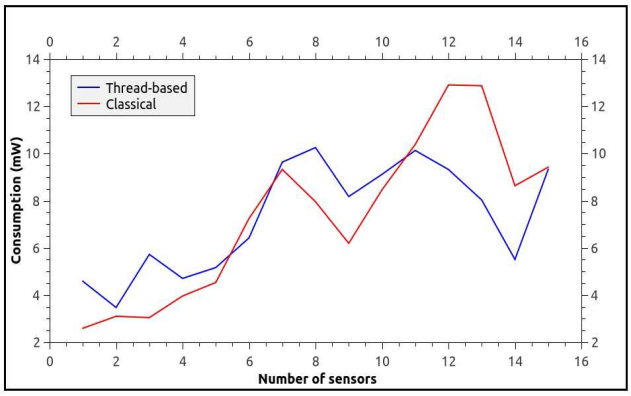

Figure 6. Comparison of the energy consumption on 30 cycles – Classical Vs Thread-based

The low energy consumption observed (less than 10 senders) on the classical architecture is explained by the fact that each process implemented in Contiki is a protothread that have a smaller memory footprint than thread, because protothreads created in the same context share the same memory space unlike threads like shown in Figure 7.

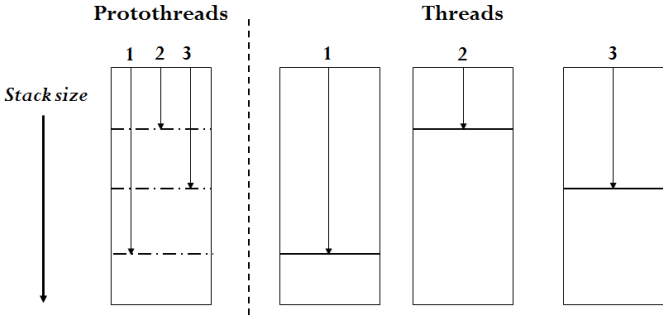

Figure 7. Comparison of the energy consumption on 30 cycles – Classical Vs Thread-based

4. The proposed approach

This section describes the proposed Multithreading Model for an Efficient Data Delivery (MMEDD). Design goals and the proposed multithreading model are presented hereinafter.

4.1 Design goals

In order to have a multithreaded architecture with a low power consumption, we focused on the characteristics of protothreads (see Figure 6). The proposal aims at providing an efficient model that has noticeably the same performance than multithreaded, which consumes less energy. Instead using several small stacks for each thread, we need to implement a share memory during run-time like in the case of the protothreads in Contiki OS.

4.2 Multithreading model

Our multithreading model operates as follows: when a node receives a packet for retransmission, it creates another process that will be responsible for retransmission. Knowing that in the Contiki system, a process is equivalent to a protothread, the first protothread *PT_recv* is responsible for listening on the communication channel. Upon reception of a request from a sender-node, if that message concerns a retransmission, the protothread *PT_recv* then creates a

second protothread *PT_send* in charge of delivering the message to the receiver-node. The proposed model is given in Figure 8.

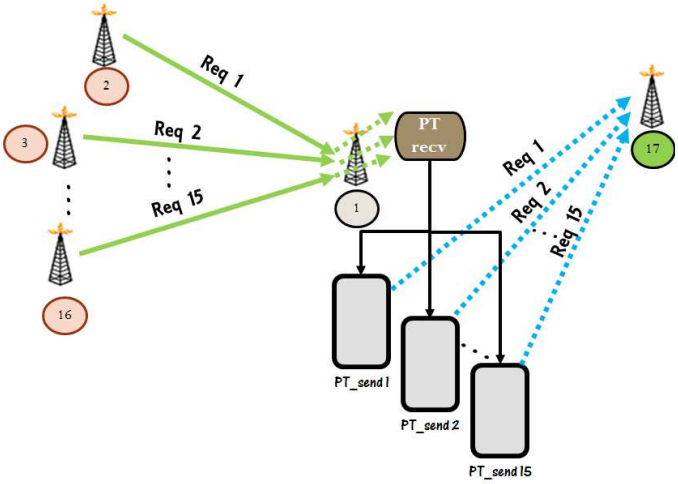

Figure 8. Multihop transfer using the MMEDD

The flow chart given in Figure 9 shows in detail the different features of the inter-node in the MMEDD architecture. At first, the node starts the listening on a communication channel with the *multihop_open()* method, the *packetbuffer* structure informs when a new arrival packet is or not dedicated to transmission. If so, the *call_back1()* method is then executed and triggers the protothread *PT_send* using the *process_start()* method and making the data delivery with the *unicast_send()* method before stopping. The protothread *PT_recv* is started from the beginning of the application.

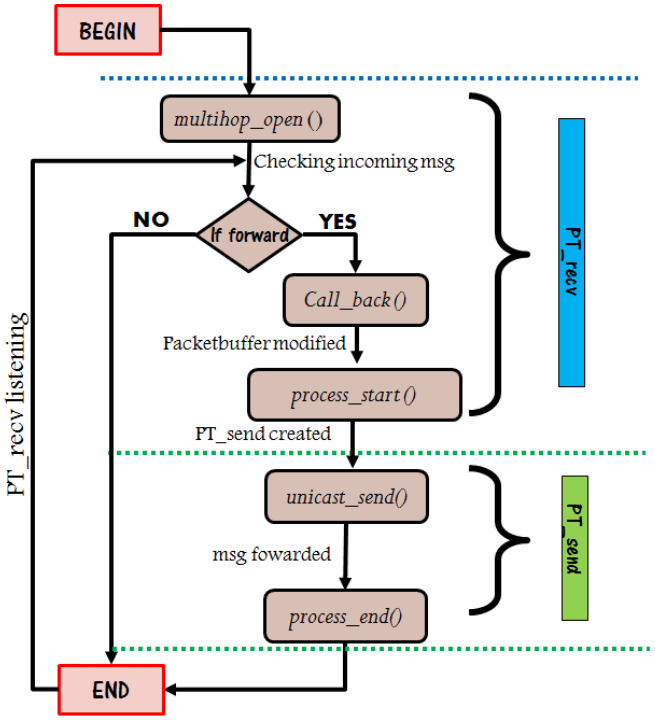

Figure 9. Architecture

For more details, the following paragraph describes the behavior of each component of the flowchart of Figure 9.

At the beginning, when a node is started, the OS creates a protothread *PT_recv* which opens a connection via a channel such as the sockets on traditional network, by calling the with *multihop_open()* function. Opening a connection on Rime needs to define a callback function and a channel number (numbers on 16 bits, numbers less than 127 are kernel reserved). Only two nodes using the same channel can communicate each other. It is important to note that any node can be an inter-node. When an inter-node receives a packet through the *packetbuffer* structure, it checks the value of the field PACKETBUF_ADDR_ERECEIVER (address of the final receiver) and compares it with its own address. Let us remind that this is achieved by *multihop_open()*. Two cases are possible:

- The final receiver is the inter-node: No need to forward the packet, the packet has reached its destination, the *PT_send* is not called and PT_recv continues listening for incoming message.
- The final receiver and the inter-node are different: The ingoing packet has to be forwarded, the *callback()* function is therefore executed. This means that *PT-recv* has to make a forward. To achieve that, the *packetbuffer* is modified by changing the value of the field PACKETBUF_ADDR_SENDER (immediate sender address is changed to inter-node address) and the also the value of the field PACKETBUF_ADDR_RECEIVER (immediate receiver becomes the final receiver). Immediately after, the *process_start()* function is called. This function executes *process_post_synch()* which is in charge to create *PT_send* while assigning it a synchronous event. At that step, the *PT_recv* steel listening on the open channel, while the protothread *PT_send* sends the modified packetbuffer to the final receiver using the *unicast_send()* function. *PT_send* is immediately destroyed at the end of its task by the *process_end()* function in order to reduce energy wastage of the inter-node.

At the final step (END) *PT_recv* returns on listening for incoming messages.

5. Simulation and Discussion

In this section, we present the experimental results of the MMEDD architecture by considering the same conditions described in Section 3.2.1. Experiments were conducted in order to evaluate message reception rate and the energy consumption in classical, thread-based and MMEDD architectures.

5.1 Message reception rate

Table 4 presents the amount of messages received by the inter-node in classical, thread-based and MMEDD architectures.

Table 4. Amount of received message on 30 cycles – Classical - Thread-based - MMEDD

Senders	Msg-Send	Classical	Threads	MMEDD
1	30	30	30	30
2	60	42	56	50
3	90	57	70	66
4	120	90	94	94
5	150	90	120	110
6	180	134	151	157
7	210	146	165	172
8	240	177	179	192
9	270	179	202	198
10	300	172	205	220
11	330	201	213	208
12	360	198	218	203
13	390	189	177	186
14	420	169	237	187
15	450	188	218	218

5.1.1 Classical vs MMEDD

In this test, we compared the number of received messages both in the classical and MMEDD architectures. Data from Table IV show that the MMEDD architecture receives more messages than the classical architecture. This result is clearly plotted in Figure 10. Indeed, our proposal observes a reception rate of 70.95% against 63.74% for the classical architecture, i.e., a gain of 7.21%. The visible gain is allowed by the fact that, the MMEDD quickly deals more with different requests received by its multithreaded structure therefore faster transfer packets to the recipient.

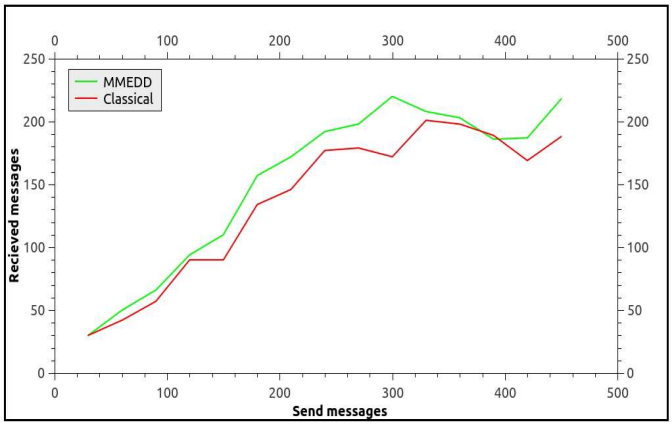

Figure 10. Comparison of number of received message on 30 cycles – Classical Vs MMEDD

5.1.2 Thread vs MMEDD

In Table 4, when considering the number of received messages between the thread-based architecture and the MMEDD, we found that these two architectures have almost identical performances (72.33% for the thread-based and 70.95% for the MMEDD). The intertwining between the two curves observed in Figure 11 shows that the thread-based and the MMEDD architectures have slightly the same reception rate (difference of 1.38%). This result is explained by the fact that these two architectures are based on multithreaded structures.

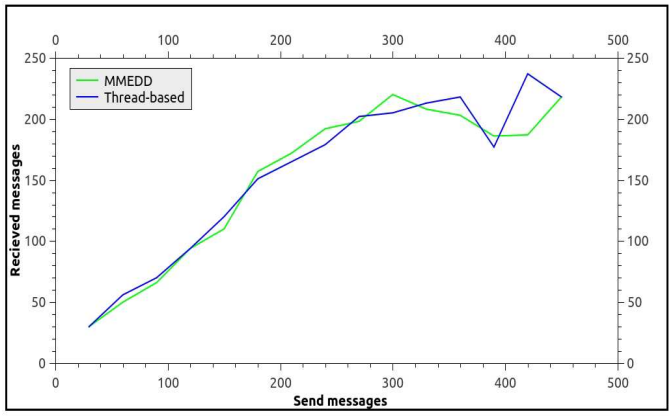

Figure 11. Comparison of number of received message on 30 cycles – Thread-based Vs MMEDD

5.2 Energy consumption

The analysis of the energy consumption in the MMEDD architecture was made by considering the energy model given in Equation 1. The obtained values for the energy consumed are listed in Table 5.

Table 5. Energy consumption on 30 cycles – Classical – Thread - MMEDD

Senders	Classical	Threads	MMEDD
1	2.5956	4.5877	1.6882
2	3.1021	3.4757	2.8172
3	3.0470	5.7267	3.6935
4	3.9621	4.70905	3.8473
5	4.5368	5.1678	3.8130
6	7.2492	6.4264	6.2966
7	9.3301	9.6382	8.6741
8	7.9583	10.2538	6.6649
9	6.1979	8.1779	5.2931
10	8.4618	9.1186	11.5380
11	10.3861	10.1320	5.2282
12	12.9091	9.3255	9.3529
13	12.8760	8.0336	10.4199
14	8.6349	5.5132	7.4984
15	9.4236	9.3250	9.8375

5.2.1 Classical vs MMEDD

In Table 5 the values of energy consumption in mW between classical and MMEDD architectures are presented. It can be seen that, in most cases the MMEDD observes lower energy consumption, for an average energy consumption of 6.4441 mW for MMEDD and 7.3780 mW for classical architecture. It can be observed that the MMEDD less consumes energy than the classical architecture. This can be explained by the fact that the two architectures implement protothreads that use less memory because sharing a same memory space, but MMEDD delivers messages more quickly. Figure 12 provides the plots of these results.

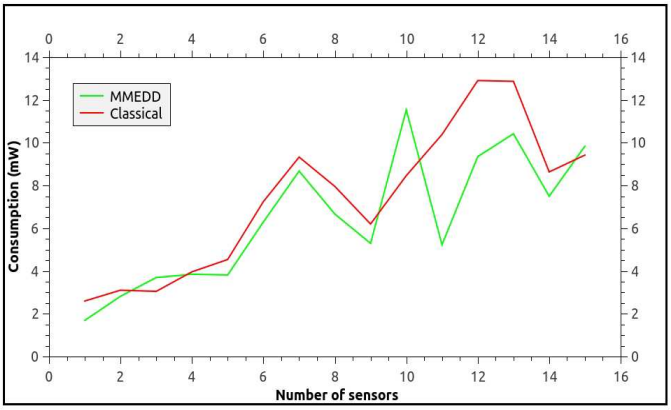

Figure 12. Comparison of energy consumption on 30 cycles – Classical Vs MMEDD

5.2.2 Thread vs MMEDD

Table 5 also presents the values of the energy consumption between the thread-based and the MMEDD architectures. We note that the MMEDD consumes in general less energy than the thread-based. The average energy consumption of the thread-based is 7.3074 mW and 6.4441 mW for the MMEDD architecture. Thus, the thread-based architecture consumes on average 0.8633 mW more than the MMEDD architecture. The different values of energy consumption are plotted on Figure 13. It can be observed that the energy consumption in the MMEDD architecture is generally less than the consumption enabled by the thread-based architecture.

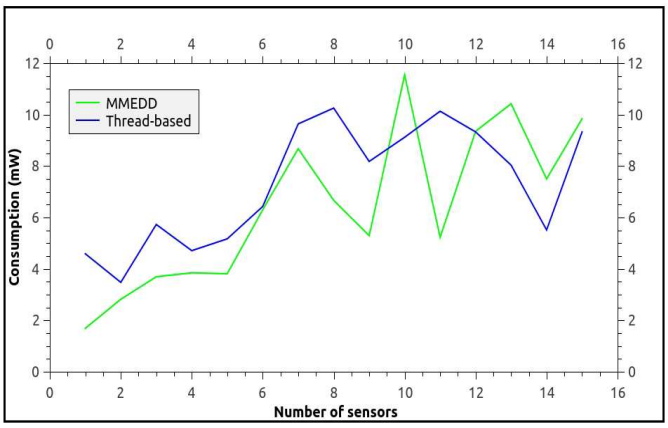

Figure 13. Comparison of energy consumption on 30 cycles – Thread-based Vs MMEDD

5.3 Message reception rate / Energy consumption

This subsection presents the efficiency of the different architectures namely classical, thread-based and MMEDD, by measuring the ratio between the message reception rate and the energy consumption. The formula given in Equation 2 performs the reception rate over the energy consumption.

$$T_{M/E} = \frac{\sum_{i=1}^{n} \frac{r_i}{e_i}}{n} \qquad (2)$$

Where:

- $T_{M/E}$ represents the average of message reception rate over the energy consumption;
- r_i is the amount of received messages with i senders
- e_i represents the energy consumption of inter-node with i senders
- n is the number of senders (15)

Table 6 gives the values for each number of senders, the ratio between reception message rate and the associated energy consumption (r_i/e_i).

Table 6. Message reception rate over the energy consumption on 30 cycles – Classical – Thread - MMEDD

Senders	Classical	Threads	MMEDD
1	11,5580	6,5392	17,7704
2	13,5392	16,1118	17,7481
3	18,7069	12,2234	17,8692
4	22,7152	19,9615	24,4327
5	19,8377	23,2207	28,8486
6	18,4847	23,4968	24,9340
7	15,6482	17,1193	19,8291
8	22,2409	17,4569	28,8076
9	28,8807	24,7007	37,4071
10	20,3266	22,4815	19,0674
11	19,3527	21,0225	39,7842
12	15,3380	23,3767	21,7044
13	14,6784	22,0324	17,8504
14	19,5717	42,9877	24,9386
15	19,9499	23,3780	22,1601

Results from Table 6 show that MMEDD has a better ratio *message reception rate/energy consumption* than thread-based and classical architectures. The average ratio *message reception rate/energy consumption* for classical, thread-based and MMEDD architectures is represented by diagrams in Figure 14. It can be observed that MMEDD performed better than the others.

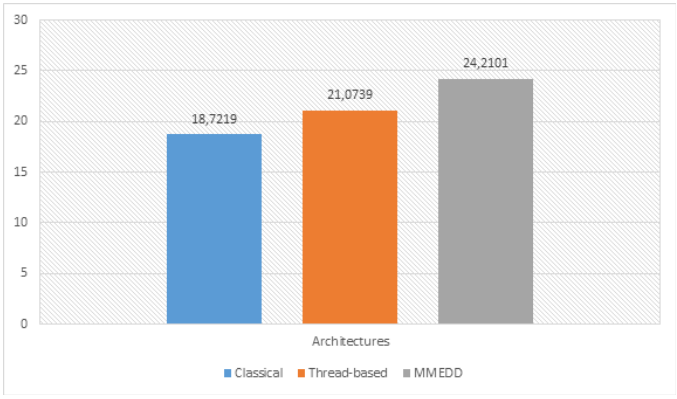

Figure 14. Comparison of the message reception rate over energy consumption on 30 cycles – Classical - Thread based – MMEDD

6. Conclusions and Future Works

In this paper, we defined a new concept of multithreading for data delivery in WSNs. To achieve this, we used the multihops sending to simulate the multitasking in the Contiki operating system. First, experiments were conducted on classical and thread-based architectures. The results obtained during these experiments have shown that the thread-based architecture has better performance in data processing speed compared to the classical architecture (*72.33%* and *63.74%* respectively). In the case of energy analysis, the thread-based architecture is greedier than the classical architecture. However, when the multitasking is important, the energy consumption of the thread-based architecture joins the one obtained in the classical architecture. According to these results, we modelled the MMEDD architecture that is a thread-based architecture.

Furthermore, instead of threads that use more memory, we integrate protothreads. To achieve this, two protothreads were designed. The first is responsible for reception and the second is responsible for transmission of packets to the destination. The large experiments conducted show that the proposed MMEDD architecture has a better power message delivering than the classical (*7.21%* more) and substantially equal to the thread-based architecture. The used protothreads that are lightweight processes implemented in Contiki favoured a lower consumption than classical and thread-based architectures. Finally, the results show that MMEDD provides better ratio *message reception rate/energy consumption* than classical and thread-based architectures.

Future works will investigate the use of other criteria such as the duty cycling, the loss of local variables by protothreads after crash.

7. Acknowledgement

The authors would like to thank the editor and the anonymous reviewers for their insightful comments and remarks that lead to improvement of the content and presentation of the paper.

References

[1] A. A. A. Ari, A. Gueroui, N. Labraoui, and B. O. Yenke, “Concepts and evolution of research in the field of wireless sensor networks,” International Journal of Computer Networks & Communications, Vol. 7, No. 1, pp. 81–98, 2015.

[2] W. Dargie and C. Poellabauer, “Fundamentals of wireless sensor networks: theory and practice”. John Wiley & Sons, 2010.

[3] E. T. Fute and E. Tonye, “Modelling and self-organizing in mobile wireless sensor networks: Application to fire detection,” International Journal of Applied Information Systems, IJAIS, New York, USA, Vol. 5, No. 3, 2013.

[4] C. Intanagonwiwat, R. Govindan, and D. Estrin, “Directed diffusion: a scalable and robust communication paradigm for sensor networks,” in Proceedings of the 6th annual international conference on Mobile computing and networking. ACM, pp. 56–67, 2000.

[5] R. de Fréin, R. Loomba, and B. Jennings, “Selecting energy efficient cluster-head trajectories for collaborative mobile sensing,” in IEEE Global Communications Conference (GLOBECOM). IEEE, pp. 1–7, 2015.

[6] A. A. A. Ari, B. O. Yenke, N. Labraoui, and A. Gueroui, “Energy efficient clustering algorithm for wireless sensor networks using the ABC metaheuristic,” in 2016 International Conference on Computer Communication and Informatics (ICCCI), Coimbatore, Tamilnadu, India, IEEE, pp. 1–6, 2016.

[7] C. Titouna, M. Aliouat, and M. Gueroui, “FDS: fault detection scheme for wireless sensor networks,” Wireless Personal Communications, Vol. 86, No. 2, pp. 549–562, 2016.

[8] N. Labraoui, M. Gueroui, and L. Sekhri, “On-off attacks mitigation against trust systems in wireless sensor networks,” Computer Science and Its Applications, Springer, pp. 406–415, 2015.

[9] J. P. L. Licudis, J. C. Abdala, G. G. Riva, and J. M. Finochietto, “Flexible prototyping for ad hoc wireless sensor network protocols,” in 40th Jornadas Argentinas de Informtica (JAIIO), Cordoba, Argentina, 2011.

[10] L. Saraswat and P. S. Yadav, “A comparative analysis of wireless sensor network operating systems,” International Journal of Engineering and Technoscience, Vol. 1, No. 1, pp. 41–47, 2010.

[11] A. Dunkels, O. Schmidt, T. Voigt, and M. Ali, “Protothreads: simplifying event-driven programming of memory-constrained embedded systems,” in Proceedings of the 4th international conference on Embedded networked sensor systems, ACM, pp. 29–42, 2006.

[12] A. Dunkels, B. Grönvall, and T. Voigt, “Contiki-a lightweight and flexible operating system for tiny networked sensors,” in Local Computer Networks, 29th Annual IEEE International Conference, pp. 455–462, 2004.

[13] G. Wang and G. Cho “Securing Cluster Formation and Cluster Head Elections in Wireless Sensor Networks”, International Journal of Communication Networks and Information Security (IJCNIS) Vol. 6, No. 1, pp. 70-87, 2014.

[14] A. A. A. Ari, B. O. Yenke, N. Labraoui, I. Damakoa, A. Gueroui, “A power efficient cluster-based routing algorithm for wireless sensor networks: Honeybees swarm intelligence based approach”, Journal of Network and Computer Applications, Elsevier, Vol. 69, pp. 77-97, 2016.

[15] N. Labraoui, M. Gueroui, M. Aliouat, and J. Petit, “Reactive and adaptive monitoring to secure aggregation in wireless sensor networks,” Telecommunication systems, Vol. 54, No. 1, pp. 3–17, 2013.

[16] A. A. A. Ari, A. Gueroui, N. Labraoui, B. O. Yenke, C. Titouna and I. Damakoa, “Adaptive Scheme for Collaborative Mobile Sensing in Wireless Sensor Networks: Bacterial Foraging Optimization approach,” in Personal, Indoor, and Mobile Radio Communications (PIMRC), 2016 IEEE 27th Annual International Symposium, Valencia, Spain, IEEE pp.1-6, 2016

[17] S. Sahraoui and S. Bouam, “Secure Routing Optimization in Hierarchical Cluster-Based Wireless Sensor Networks”, International Journal of Communication Networks and Information Security (IJCNIS) Vol. 5, No. 3, pp. 178- 185, 2013.

[18] J. Hill, R. Szewczyk, A. Woo, S. Hollar, D. Culler, and K. Pister, “System architecture directions for networked sensors,” ACM SIGOPS operating systems review, ACM, Vol. 34, No. 5, pp. 93–104, 2000.

[19] T. Reusing, “Comparison of operating systems tinyos and contiki,” Sens. Nodes-Operation, Netw. Appli.(SN), Vol. 7, 2012.

[20] C. -C. Han, R. Kumar, R. Shea, E. Kohler, and M. Srivastava, “A dynamic operating system for sensor nodes,” in Proceedings of the 3rd international conference on Mobile systems, applications, and services. ACM, pp. 163–176, 2005.

[21] H. Cha, S. Choi, I. Jung, H. Kim, H. Shin, J. Yoo, and C. Yoon, “Retos: resilient, expandable, and threaded operating system for wireless sensor networks,” in Information Processing in Sensor Networks, IPSN 2007. 6th International Symposium, IEEE, pp. 148–157, 2007.

[22] V. Vanitha, V. Palanisamy, N. Johnson, and G. Aravindhbabu, “Liteos based extended service oriented architecture for wireless sensor net-works,” International Journal of Computer and Electrical Engineering, Vol. 2, No. 3, p. 432-436, 2010.

[23] A. Eswaran, A. Rowe, and R. Rajkumar, “Nano-rk: an energy-aware resource-centric rtos for sensor networks,” in Real-Time Systems Symposium, RTSS 2005, 26th IEEE International, pp. 1—10, 2005.

[24] X. Liu, K. M. Hou, C. de Vaulx, J. Xu, J. Yang, H. Zhou, H. Shi, and P. Zhou, “Memory and energy optimization strategies for multithreaded operating system on the resource-constrained wireless sensor node”, Sensors, Vol. 15, No. 1, pp. 22–48, 2014.

[25] A. Sehgal, “Using the contiki cooja simulator,” Computer Science, Jacobs University Bremen Campus Ring, Technical Report, 2013.