Improving Network Efficiency for Competitive Data Fusion in Wireless Sensor Networks using Genetic Machine Learning Algorithm

Rahul Balamurugan

(Group 25)

*Abstract*— **In dense wireless sensor networks (WSNs), unpredictable failures or changes may occur when human intervention may not be possible. In such networks, self-management and redundancy is required for robustness and to ensure quality of information gathered. Applications in the modules may have conflicting objectives, which makes management more complex. This project is an OMNET++ implementation of a Genetic Machine Learning Algorithm (GMLA) proposed in another paper to improve the efficiency of an IEEE 802.15.4 WSN. The report contains the detailed implementation using the INET framework with the detailed custom classes required to reproduce the work given in the attached appendices. The simulation results in the original work were verified by running similar experiments with the implemented network in the project.**

1. INTRODUCTION

Wireless sensor networks (WSNs), generally composed of nodes having some number of modules (e.g., communication, application), limited processing power and finite power source, are often deployed in regions where human intervention is not possible (due to inaccessible terrain or remoteness) when the network topology changes unpredictably due to node failure, communication failures or loss and rediscovery of nodes [1]. Such changes may be caused by changing physical environments, loss of power, or simply hardware malfunctions, the probability of which occurring increases with the number of devices in the network. This leads to a need for redundancies and a level of self-management in WSNs to ensure a robust system and accurate information transmission to the end user [1]. In addition, the management, optimization and/or repair of the network often needs to happen dynamically and cater to applications with differing objectives, which adds to complexity when designing modules to provide such services.

Dense WSNs, which are networks with relatively large number of nodes per square meter, present their own challenges when it comes to management. In wireless sensor networks based on the commonly used [2] IEEE 802.15.4 standard, the MAC layer protocol avoids transmission collisions and reduces number of dropped messages or transmission failures, but only when the network is small [3]. In dense WSNs based on the same, the protocol fails in preventing congestion if implemented alone as it was not designed for such implementations. The network efficiency drop with increased density in such networks can be seen in **Figure** **4**. This shows the need for other modules to regulate the network traffic [1] in such systems. In addition, some tasks undertaken at the master node may not allow for staggered message pre-scheduling as they require the data to be sent in real-time as events are discovered and in sufficient frequency and volume that quality requirements are met. For example, data fusion tasks, especially in the case of parallel data fusion, may have changing requirements for number of messages received in a certain duration and interval. This report describes the OMNET++ [4] implementation of a classifier system based on Genetic Algorithms (GA), called Genetic Machine Learning Algorithm (GMLA) in regulating the message transmission from multiple slave nodes to a singular master node; so as to achieve higher transmission efficiency while meeting quality requirements for a parallel data fusion task. The implementation is entirely based on the algorithm and methods described in the paper [5].

The optimization approach is based on the metric determined as ratio of messages received at the master nodes (where the data fusion happens) to the number of messages sent in a monitoring cycle. The network is trained in runtime, i.e., while it is up, which is more appropriate for WSNs as compared to more traditional optimization techniques such as the regular GA.

The objectives of the project were two-fold— one, to implement the algorithm and network as described in [5]; and two, to verify the results presented in the paper with regards to comparing the baseline implementation with GMLA by doing similar simulations. The main tasks of the project were the writing of a GMLA control mechanism in the master node application using C++, creating the other OMNET++ simulation files, and gathering (and analyzing) the results by running the simulation. The implementation is described in Section 4, and the full code of the custom applications is in Appendix A. The results are presented and described in Section 5, and the .CSV and .VEC format files of the same are in the compressed folder submitted alongside the report under ‘results’.

2. BACKGROUND

***2.1. IEEE 802.15.4***

The IEEE 802.15.4 standard [6] defines the operation of low-rate and low-power wireless personal area networks (LR-WPANs). The transmission bit rates (<250 kbps) and working frequencies (868/900 MHz) described are conducive to WSNs making it a popular standard for such applications, which can be noted from the fact that the standard forms the basis of the widely used *ZigBee* and *WirelessHART* specifications [2]. Of particular interest to this project is the MAC protocol described— the Carrier Sense Medium Access/ Collision Avoidance protocol (CSMA/CA)[3], specifically the beaconless mode.

The principle of the CSMA/CA in beaconless mode is that each device waits for the channel to clear before transmitting their message packet. To achieve this, each device in the network maintains a backoff exponent value and the number of times they are forced to backoff. “Backoff” here refers to the number of times the node tried to send a message and failed because the channel was occupied by another node. The time the node takes between each try is determined by the backoff exponent, which naturally increases exponentially between each retry. A min and max value of the backoff exponent can also be configured to limit the value. Additionally, a max backoff number value (default 4) can be configured to let the node know when to drop the message and go to the next state.

***2.2. Classifier Systems, GMLA***

Learning Classifier Systems, or simply classifier systems, in general are rule-based systems that combine search algorithms like GA with a machine learning or updating algorithm [7]. The more common usage of the term is as done in this project and the parent paper [5], is that of rule-based systems that leverage machine learning methods in solving optimization problem using Genetic Algorithms and is thus called GMLA. The rules referenced here are called classifiers, which are composed of condition-action pairs and can be used to control some system behavior by prescribing a certain action or set of actions to a detected environmental condition.

While the full description of the GMLA is given in Section 4.1.4, the basic concepts of the algorithm are introduced here. Similar to traditional GA, the GMLA does evaluation, selection, evolution (selection, crossover, and mutation) and replacement of poorly performing members of the population with new members. Where it is different is that the classifiers which take the place of creatures or solutions in the population (in GA) in evaluation at any given instant are chosen by consulting with the environment (getting current condition) and evolution operations are done on the chosen classifiers after certain conditions are met, generally after a set duration has elapsed.

The machine learning aspect comes from how better performing classifiers are differentiated from bad ones— the system used is an apportionment component, symbolically called a bank, to which each classifier must pay a tax from its fitness value, called budget, when it is chosen. The ones with good performance are rewarded with the amount in the bank in later iterations.

Also to be noted is that the GMLA, in contrast to regular GA and other optimization techniques, works online. That is to say, it performs the selection of classifiers, evaluation, and evolution— all during the execution itself. This implies that there will be some fluctuations in the network quality until an optimum is reached and the system stabilizes. As can be noted in [5] and in the simulation results (**Figure** \*), a “Learning cycle” of only ~30 s was sufficient after which the system stabilizes. This speed of settling or adapting to a new situation is one of the hallmarks of this algorithm.

***2.3. Competitive Data Fusion***

Data fusion refers to the meaningful aggregation of data from various sources or a single source over a period of time at a sink. Competitive or parallel data fusion in particular refers to receiving the same data from multiple sources redundantly to ensure robustness and accuracy.

Although the data fusion is mentioned in the title, and indeed forms the basis for the metrics used in the project, data fusion is not performed in the project in the sense that no meaningful messages are transmitted between the nodes in the OMNET++ simulation. On the contrary, an arbitrary threshold of number of messages received is assumed to be necessary for successful data fusion, and this acts as a constraint for the GMLA control mechanism.

3. RELATED WORK

Chart, bar chart, box and whisker chart

Description automatically generated

**Figure 1**: Example monitoring cycles, with one slave node.

The project is based entirely on the paper [5] by Pinto *et al*., and therefore is the solely discussed piece of peer-reviewed literature in this section. In the paper, the authors present the central idea of the control mechanism as implemented here— the Sending Probability (*SP*). Discussed further in the next section, this variable is a global variable known to all nodes in the network and updated by the master node after GMLA computation periodically. It controls the probability that the slave node will send a message or not, i.e., it reduces the number of concurrently waiting to transmit nodes while also being non-deterministic to avoid information bias. From the successful implementation in their paper, as well as in this project, controlling the *SP* value allowed for adjusting the efficiency of the network. The paper also defined two metrics— Efficiency (*Ef*) and Quality of Fusion (*QoF*) which are discussed and introduced in the following section (4.1.5). In addition, the authors show the usage of an expert phase based on the best results obtained via an initial learning phase, which again was used in the project.

Further, the authors present several interesting results. One of them being that the *QoF* which is the number of messages received at the master node before the deadline for successful data fusion, was maintained at a high level in GMLA case (desirable result), even though it was not adjusted or controlled in any way, as compared to a control experiment just implementing IEEE 802.15.4 MAC protocol. This was directly the opposite in the case of less dense networks (below 25 nodes in 100 m2). They conducted both hardware implementation and simulation in the paper and proved that the GMLA method in addition to the IEEE 802.15.4 MAC protocol was superior to both implementing the IEEE 802.15.4 directly, as well as the Gur Game— a well-documented self-management method, in dense WSNs (>65 nodes). The work also mentions some papers that use similar metrics and optimization techniques to reach various objectives and manage WSNs.

4. METHODOLOGY

***4.1. Design***

*4.1.1. Network*

The network was designed in a star configuration, with a variable number of slave nodes placed randomly, each communicating with the master node in a single hop. This was one of only two options since the IEEE 802.15.4 standard [6] only allows two configurations— star and peer-to-peer. The MAC layer protocol was the default IEEE 802.15.4 CSMA/CA in beaconless mode. As explained in [1], the star network is one of the most commonly used configurations, and since we are discussing parallel data fusion, it is the most optimal way to gather information from multiple sources at a single node in terms of management simplicity and because IEEE 802.15.4 standard specifies the use of 868/900 MHz which allows for long distance line-of-sight communication. From the simulations it was observed that at the relatively low power of 2.24 mW, the communication range was around 100 m, as shown by the blue boundary circle in the network **Figure** **2**.

The user datagram protocol (UDP) was used as the transport layer protocol since complex handshakes and assured delivery are not required in dense WSNs. The network layer was set to be IPv4 and the routing tables for static nodes were fixed. An additional case of mobile slave nodes was also designed, with the nodes moving randomly about an area much larger than the communication range to show the effect of unpredictable changing topology.

*4.1.2. Monitoring cycle*

The master and slave nodes send and receive messages periodically. Each big period, called a macro-cycle consists of the master sending a broadcast control message with the value of the sending probability *SP* to the slave nodes at the very beginning, followed by a number of micro-cycles, during which slave nodes will try to send messages to the master node. In each micro-cycle, a number of slave nodes will be actively trying to send one message each, and the number of slave nodes active in each micro-cycle is determined by the *SP*. The structure of a monitoring cycle is shown in **Figure** **1**.

A maximum delay limit of same value as a micro-cycle is set at the master node to ensure that only messages sent on time are used for the competitive data fusion task.

*4.1.3. Metrics*

To keep track of system performance and evaluate the effectiveness of the control signal *SP*, the authors in [5] defined the metrics Efficiency (*Ef*) and Quality of Fusion (*QoF*) as introduced in Section 3. *Ef* measures the rate of transmission success from slave nodes to master node and is simply the ratio of messages successfully received to the number of messages sent in one macro-cycle,

where is the estimate of messages sent in the macro-cycle,

and where is the total number of slave nodes, and is the number of micro-cycles in each macro-cycle. The value is an estimate because the *SP* is non-deterministic, meaning that there may arise some situations where the calculated *Ef* is greater than 1, which is practically not possible.

*QoF* is the average number of messages received over each macro-cycle. It is how many messages are available for the data fusion task in each micro-cycle and forms the basis for deciding if the network can successfully fulfill its responsibilities.

The main metric used in the GMLA is *Ef*, while *QoF* is used as a constraint, in that any chosen solution must result in a *QoF* above a chosen threshold. In essence, the GMLA does a trade-off between *Ef* and *QoF* by controlling the *SP*.

**Algorithm 1**: GMLA as implemented in the project

|  |  |
| --- | --- |
| 1: | In: LastEf, LastClassifier, Evolution\_Interval, BestEf |
| 2: | BestSPList, currentSP, CurrentPhase |
| 3: | Out: NewSP |
| 4: |  |
| 5: | At (t = 0), do: |
| 6: | Initialize\_population(); |
| 7: |  |
| 8: | At (t = 1, 2, ... M), do: |
| 9: | Ef = calculateEfficiency(); |
| 10: | QoF = calculateQoF(); |
| 11: | ; |
| 12: | If (), then: |
| 13: | Last\_classifier->budget += reward; |
| 14: | End if |
| 15: | If ( & QoF > QoFThreshold), then: |
| 16: | BestSPList[next]←currentSP; |
| 17: | End if |
| 18: | Condition = consultClassifierSystem(); |
| 19: | NewClassifier = selectBestClassifier(Condition); |
| 20: | NewClassifier->budget -= tax; |
| 21: |  |
| 22: | At (t = nEvolution\_Interval, n=0,1,2...), do: |
| 23: | For all Conditions, do: |
| 24: | BestParents = selectBest2Parents(); |
| 25: | mateParents(BestParents); |
| 26: | replaceWorst2Classifiers(); |
| 27: | NewMembers->budget = 100; |
| 28: |  |
| 29: | If CurrentPhase == Expert, then: |
| 30: | NewSP ← BestSPList; |
| 31: | If CurrentPhase == Learning, do: |
| 32: | NewSP = getSPValue(NewClassifier); |

*4.1.4. GMLA and control mechanism*

At the end of each macro-cycle, the master node calculates the *Ef* and *QoF* values for that cycle. In the learning phase, where the decision of new *SP* value for next cycle is determined using GMLA at the master node, this value acts as an input. With the value of the macro-cycle’s *Ef*, the change in *Ef* from the previous cycle is measured as

As introduced in Section 2.2, the GMLA is a rules-based system. The rules used are called classifiers, composed of condition-action pairs, where each part is composed of 2 segments as shown in **Table 1**. The condition part is determined from the — the sign determines whether the *Ef* decreased or increased, and forms the first segment of the condition part, and the second segment is how much it has changed. The action part is similarly composed of one segment telling the system to increase or decrease the *SP*, and another dictating how much it has to change by. Each classifier also has a budget component, which while not involved in the control aspect of the classifier, serves to determine how good or bad a classifier is for the system.

In this project (as in the paper [5]) there are 16 possible conditions, as each condition is of 4 bits— 1 for the sign, and 3 for the level. Each of the 16 conditions are equally represented in the population of classifiers, with 4 actions each also composed of 4 bits. Thus, 64 classifiers form the population that represent the initial solution space (rule base). The breakdown of the structure is given in **Table** 1. Each classifier is assigned a budget of 100 at initialization.

In the algorithm, the classifiers suitable for the next cycle are the 4 classifiers with conditions matching the value. Each time (once every macro-cycle) the GMLA checks the current condition against the conditions in the population, it is called a consult. Out of the 4, the one with the highest budget is selected as next classifier. In case multiple classifiers have the same budget, the next classifier is chosen randomly from the max valued ones. When chosen, 10% of the classifier’s budget is paid to the bank (tax).

If the value of , meaning the previous classifier (activated in previous macrocycle) was better than the current one (note that this is assuming we are at the end of a macrocycle here), then the previous classifier is rewarded with all the amount stored in the bank (reward).

After a period of time has passed, in the case of the project, 4 macro-cycles, evolution takes place. In this process, for each of the set of 4 classifiers having the same condition (16 x 4), the best performing 2 classifiers are chosen, and naturally the remaining 2 are the worst performing ones. Only considering the action part, each of the best performing classifiers act as parents for the next generation. The mating process is composed of fixed point crossover operation and a mutation operation for a random bit that can occur based on a fixed probability, here 0.01. The mating process results in 2 new children classifiers, which replace the worst performing 2 (their budget is set to default starting values) and thus the new population is formed. The full algorithm is given under **Algorithm 1**.

**Table 1**: Classifiers’ structure: condition = <C1, C2> and action = <A1, A2>. Taken directly from [5] (table 2).

A picture containing text

Description automatically generated

While the operation of the system is determined by the *SP*, the machine learning part of the GMLA is exploited in [5] to stabilize the system. This is done by separating the operation of the system into 2 phases— a learning phase and an expert phase.

In the learning phase, the *SP* decision is made based on the classifiers chosen using the GMLA. After the algorithm is run for a sufficient duration, or a number of good values of *SP* (good values are those for which *QoF* > Threshold) are found, the system goes into the expert phase, where the *SP* is determined from the list of best SPs and the GMLA does not take part in the decision process.

When the best values of *SP* are stored in the learning phase, each value is assigned a score of 1. This score is increased if the injection of the value into the system leads to *QoF* > Threshold and decreased by 1 if not. When a value’s score reaches 0, it is discarded from the list. This mechanism allows the system to settle at a *SP* that results in adequate *QoF*. With regards to the *QoF* threshold, it can be set arbitrarily depending on the requirements of the fusion task.

***4.2. Implementation*—**

The implementation of the design was done using the INET 4 framework [8] available in OMNET++. All the network components used to model the system were used as available in the framework, except for the applications, which also contain the GMLA, further described in Section 4.2.3 below.

*4.2.1. Network*

Two networks were implemented— one for the static nodes case, where the slave nodes were placed randomly inside a square of 100 m x 100 m centered around the master node, visualized in **Figure** **2**; and the second one was for the mobile case where the initial positions of the slave nodes were in a square of 1000 m x 1000 m again centered around the master node. The second network is shown in **Figure** **3**. The nodes were all *StandardHost* modules with the number of wireless cards set to one.

The wireless interface was chosen to be the *Ieee802154NarrowbandInterface* [9], which resulted in default IEEE 802.15.4 physical (narrowband) and MAC (CSMA) specifications. In the INET framework, the IEEE 802.15.4 model default for narrowband scalar radio uses the breakpoint pathloss model in computations, with default breakpoint at 8 m.

The network layer configuration was set to be done semi-automatically using the IPv4 configurator, with routes and address patterns specified using a .xml file.

*4.2.2. Applications*

While the network was based on standard components available in the INET framework, the more complex and custom requirements for the *SP* signal sending and reception, and for the GMLA meant that custom applications had to be designed for the task. Two applications were developed from the *UdpBasicBurst* [10] class for this purpose, one for the master node module and one for the slave node modules.

Diagram

Description automatically generated

**Figure 2**: Static nodes configuration network with 35 nodes

Summarizing the requirements for the master node— all messages from the slave nodes needed to be received every micro-cycle, while discarding those that arrive after the deadline; and at the start of each macro-cycle, the master node should emit a new value of *SP*, computed using the GMLA.

Doing the same for the slave nodes— each node should try to send a message with a probability controlled by the *SP* parameter at each micro-cycle start except for the first one of every macro-cycle, where they should wait for the master node’s control signal.

The *UdpBasicBurst* application already had the capabilities to act as both a packet source and sink, with configurable sending intervals, a sleep interval, and a burst duration. The burst duration determines the time when the application acts as a packet source, with messages to be sent after every sending interval, and the sleep duration determines the time to the next burst. This was the reason the application was chosen as the base class for the slave node application, henceforth called SlaveApp. The major addition that needed to be done with regard to message generation was in the burst generation function, where a random (Bernoulli) number generator had to be added to decide whether the module will send in that micro-cycle or not.

With regard to the master node application, henceforth called MasterApp, the major additions to the base class were the functions to enable the GMLA. The full class and function documentation is in Appendix A. Some important functionalities added are the *controlSP* function to do the GMLA, get new *SP* values and send them to the slave nodes and a random integer generator that generates an integer sampled uniformly from a uniform distribution in given range, and modified the random seed input so that the values would be pseudo-random (reproducible) and yet non-deterministic enough to suit GA needs. The seed modification with each access idea and the method of modifying the seed were taken from the C++ library in [11].

A picture containing diagram

Description automatically generated

**Figure 3**: Mobile nodes configuration network, mid-run, with 125 nodes. The slave nodes are deployed in a 1000 x 1000 m region.

One specific functionality that warrants mention outside the documentation is how the transmission of the *SP* value to the slave nodes at the start of every macro-cycle was achieved. Although this seems to be a basic need for network simulation, it was not simple at all to understand how to add the ability to transfer the value across modules, as it cannot be done using messages to the best of my knowledge. The other avenue, which worked and was implemented in the project, consisted of dynamic casting of a pointer from the MasterApp class (function *controlSP*) to access a method in the SlaveApp class that only changed an integer value in SlaveApp (*SP* stored there). The method kept throwing errors, because of the requirement in OMNET++ to keep the executor aware of which module is being accessed at all times in the simulation [12], which was enabled by the standard OMNET++ *Enter\_Method\_Silent()* function.

With regard to the GMLA implementation in the MasterApp, the classifier was taken as a C++ struct, with an integer *condition*, integer *action*, and double *budget*. Since the conditions are simply all 16 permutations of 4 bits, they were represented by 0-15, and the actions were then randomly chosen 4 at a time at initialization from 0-15. This formed 64 classifiers, 4 for each condition. The crossover and mutation operations were bit operations performed on these integer values.

5. SIMULATION AND RESULTS

This section presents the results of experiments run on OMNET++ with the setup described in the previous section. The experimental parameters are roughly similar to those in the paper [5]. To show the effect of adding the GMLA control mechanism into the system, two sets of e experiments were conducted— first, simulating the network only relying on IEEE 802.15.4 CSMA/CA and measuring the *Ef* and *QoF* for different numbers of nodes (65, 125, 185, 250 and 310) as baseline or control set; and second, simulating the network with the master and slave nodes containing their respective custom applications (description in previous section) with GMLA and measuring the stabilized (expert phase) *Ef* and *QoF* for the same numbers of slave nodes as the IEEE 802.15.4 experiments. All simulations were run for 60 seconds total.

The simulation results were exported from OMNET++ as .CSV files and analyzed using python (Jupyter Notebook). The results are submitted inside the compressed folder under ‘results’, and all the plots generated are included in Appendix C as well.

***5.1. IEEE 802.15.4***

*5.1.1. Simulation Parameters:*

As the first set of experiments was meant to be a baseline for the GMLA implementation, the applications in the slave nodes were set to be *UdpBasicApp* applications as available in INET, and the application in the master node set to be a *UdpSink*. The sending interval parameter in the slave node applications was set to 1 s, at which time all slave nodes will try to send a message each to the master node. Since the clock time is assumed to be globally synchronized, this happens at the exact same time.

The mobility model of the nodes in the mobile case was set to *GaussMarkovMobility* with default parameters, which can be observed in the ‘omnetpp.ini’ file under [config Mobile]. Note that no extra additions to the network were made to the mobile case, except for not allowing the IPv4 configurator to make the routes static.

*5.1.2. Results*

The *Ef* and *QoF* values for the IEEE 802.15.4 alone case were computed in python after the fact rather than during execution, as they remain at an average value throughout the execution, which can be seen in Appendix C (figure \*). The plot was generated by using the GMLA network and keeping the *SP* constant at 100%.

From the results in **Figure** **4** of the static node configuration with different numbers of nodes, it can be seen that in the IEEE 802.15.4 case, the efficiency of the network decreases sharply with increase in number of nodes. This was exactly as expected, since the goal of the project is to improve the efficiency at higher number of nodes.

In the case of mobile slave nodes as can be seen in **Figure** 5, however, it can be seen that only after () nodes does the efficiency increase, and after () nodes, the efficiency again decreases, which is the same pattern observed in the GMLA case as well.

***5.2. GMLA***

*5.2.1. Simulation Parameters*

The length of one macro-cycle in the simulation was set to 1s, with each micro-cycle of 0.2 s (simulation time). In the learning phase, where evolution and learning occur, the evolution was set to take place every 3 macro-cycles, or every 15 micro-cycles (3 s). This was exactly as described in the original paper [5], with some liberty taken in setting 0.2 s for each micro-cycle (the duration was not mentioned).

The learning phase was set to end and transition to the expert phase after 30 macro-cycles or if 20 best *SP* values are stored. The *QoF* threshold was set to 9. This was assumed to be the same as the settings in the original paper by looking at the point of stabilization in their results, as this was not mentioned in the paper. The QoF threshold in particular was arbitrary, as no actual parallel fusion task is completed and if such a task were to be implemented at the master node, the requirements of the task could be met by adjusting the QoF threshold.

The budget for each new classifier after evolution as well as the initialized population was set to 100. The scores for each new best *SP* value stored was set to 1.

*5.2.2. Results*

The results for this section naturally differ from the results in the original paper in exactness, but are similar in trend, because the GMLA is non-deterministic, and their implementation definitely differs from the one in the project with regards to how that randomness was generated even if it was only pseudo-random. In addition, as mentioned in the discussion (Section 5.2.3), it is unknown whether the authors in the original paper consider the total average of *Ef* over the entire simulation or only over the expert phase. In the project, the total average over 60 s of simulation was taken, as the values were more in line with the results in the paper.

From **Figure** **6** and **Figure 7**, it can be observed that the GMLA in addition to CSMA makes the network much more efficient than only having CSMA at higher densities. Under 65 nodes, there was not much difference between the 2 implementations. However, at 125 nodes and beyond, the GMLA increased the *Ef* by ~100%.

The *QoF*, on the other hand, was low compared to that achieved by IEEE 802.15.4 (alone) at number of nodes 65 and lower. At higher numbers, the results are better in the GMLA as the implementation of a *QoF* threshold of 9 ensured only *SP* values resulting in an adequate *QoF* were stored. As noted earlier, the *QoF* achieved is not indicative of the limits of the algorithm as the *QoF* is simply due to an arbitrary threshold being set.

From the **Figure** **8** showing the execution of the GMLA network having 125 slave nodes, the volatile and random trajectory of the *Ef* in the learning phase (<30 s)and the clear stabilization (>30 s) can be noticed. This matches closely with the results in [5], making sure that the methods worked in the same way.

***5.3. Discussion on the simulation***

Some parts of the implementation in the project will differ from the description in [5], since it is either not elaborated, mentioned or vague in meaning. At these points, the assumptions have been made based on logical deductions, and from the simulation results mentioned in the paper. Such points are mentioned in this section, along with some insights garnered during the project.

Of some doubt in the authors’ implementation of the GMLA in the paper was the methods of GA evolution used, as the authors were frustratingly vague at certain points; one of them being the type of crossover operation they used— either fixed point or randomly chosen. As no special note was made, fixed point was assumed in our implementation. Another major point where assumptions had to be made was the duration of the learning phase, which had to be guessed from one of the figures (Fig. 13 in [5]) in the paper to be 30 macrocycles. The assumption may be false depending on whether the learning phase length was variable or fixed, and whether the algorithm went to the expert phase early.

Another point of contention was what value of *Ef* was recorded for each node configuration— since the GMLA has an adjustment phase, the actual values should be an average over the expert phase operation. On the other hand, comparing the simulation results achieved in the project and their numbers, it appears they have taken an average over the entire time period of around 50 macrocycles. In our particular case that would be the first 50 s worth of data. However, since this was not mentioned anywhere in the paper, the results naturally differ from that in the project. Speaking of the *Ef*, it should also be noted (and was mentioned in Section 4.1.3) that the number of messages sent during each macrocycle was but an estimate, and the metric could have computed values beyond 1, which is naturally not a very accurate representation of the actual system state. It is therefore more meaningful to look at the average rather than taking a single value as representative, such as the maximum.

Apart from this was a doubt about the nature of evolution in the proposed GMLA algorithm in the original paper: Did all 4 classifiers for every condition (16 conditions) take part in the evolution process or did only the 4 classifiers satisfying the consulted environmental condition (change in efficiency) take part? While logically, the former concept makes more sense since there would be no chance of some classifiers not going through evolution at all, the authors in the paper mention that only 2 classifiers are replaced every time evolution takes place, which seems to suggest that only the classifiers satisfying the environmental conditions were considered. In this project, the evolution considered was the latter, as mentioned in Section 4.1.4.

In just one line in the text, the authors mention that the GMLA adjusts the value of *SP* upwards if it becomes too low:

“...However, if *SP* becomes too small, GMLA detects it and tries to reduce it.”

– [5] Section 5.2.1, para 3.

This was the only indication that something was implemented to avoid the *SP* going down to near 0 and staying there, as that would in essence maximize *Ef*. While the *QoF* threshold should avoid this happening, in practice, without such a mechanism added to the application, the *SP* value does indeed dip to near zero, and stays there during the learning phase. That is why in the code (in this project), a condition was added that if at any point the *SP* decreases below 5(%), it would be adjusted upwards by 40(%) additive, as it was observed that at that value (in fact all values of *SP* >30), the algorithm worked very well. In fact, that is possibly one of the reasons why the GMLA works much better for higher number of nodes compared to few nodes— because the effect of low *SP* values is not as pronounced on the *Ef* when the number of nodes total is high.

And lastly, because the simulation program for 400 nodes and above kept hanging in Express mode, and the time taken to run the program for such dense networks was prohibitive, all the results in the original paper could not be verified. Nonetheless, the results of the simulation in the project were successful in their objectives of verifying that the GMLA method led to better *Ef* than the barebones IEEE 802.15.4 in dense WSNs.

6. CONCLUSION

The project, based on the paper authored by Pinto et al. [5], was successful in the OMNET++ implementation of the GMLA described in the paper. The simulation results for the IEEE 802.15.4 and GMLA added networks were verified by independently simulating similar experiments. The general takeaway from the project is that some additional self-management functionality should be added to dense WSNs based on the IEEE 802.15.4 standard for parallel data fusion and similar tasks where many sources may be trying to send messages simultaneously.

While the results of the simulations were on par with the reported results in the original paper for number of nodes 65, 125 and 250, the limitations on the hardware and time available made the simulations for higher number of nodes (>400) limited the ability to corroborate the results for such large numbers. More simulations may present interesting results for those interested.

One improvement that could be made on the implementation in the project include a more realistic way of sending the SP control signal periodically from the master to the slave nodes, possibly using signals and listeners [] as mentioned in the OMNET++ user guide. Also, improvements on randomness could help improve the GMLA performance and convergence, along with well-tuned parameters for length of learning cycle and evolution intervals.

REFERENCES

[1] K. Sohraby, D. Minoli, and T. Znati, *Wireless sensor networks: technology, protocols, and applications*. Hoboken, NJ: Wiley, 2007, pp. 1-20.

[2] A. G. Ramonet and T. Noguchi, "IEEE 802.15.4 Historical Evolution and Trends," 2019 21st International Conference on Advanced Communication Technology (ICACT), 2019, pp. 351-359, doi: 10.23919/ICACT.2019.8702040.

[3] R. Srivastava and A. Kumar, "Performance analysis of beacon-less IEEE 802.15.4 multi-hop networks," 2012 Fourth International Conference on Communication Systems and Networks (COMSNETS 2012), 2012, pp. 1-10, doi: 10.1109/COMSNETS.2012.6151310.

[4] A. Varga, "Using the OMNeT++ discrete event simulation system in education," in IEEE Transactions on Education, vol. 42, no. 4, pp. 11 pp.-, Nov. 1999, doi: 10.1109/13.804564.

[5] Pinto, A.R., Montez, C., Araujo, G., Vasquez, F., and Portugal, P. (2014, January). “An Approach to Implement data fusion techniques in wireless sensor networks using genetic machine learning algorithms.” Information Fusion, vol. 15, pp. 90-101. [Online. Retrieved from: <https://doi.org/10.1016/j.inffus.2013.05.003> on 20 June 2021].

[6] "IEEE Standard for Local and metropolitan area networks--Part 15.4: Low-Rate Wireless Personal Area Networks (LR-WPANs)," in IEEE Std 802.15.4-2011 (Revision of IEEE Std 802.15.4-2006) , vol., no., pp.1-314, 5 Sept. 2011, doi: 10.1109/IEEESTD.2011.6012487.

[7] R. J. Urbanowicz and J. H. Moore, "Learning Classifier Systems: A Complete Introduction, Review, and Roadmap", Journal of Artificial Evolution and Applications, vol. 2009, <https://doi.org/10.1155/2009/736398>.

[8] “User’s Guide” INET framework v.4.2.0. <https://inet.omnetpp.org/docs/users-guide/index.html>.

[9] “Ieee802154NarrowbandInterface” in API Documentation, INET Framework v.4.2.0. <https://doc.omnetpp.org/inet/api-current/neddoc/inet.linklayer.ieee802154.Ieee802154NarrowbandInterface.html>.

[10] “UdpBasicBurst” in API Documentation, INET Framework v.4.2.0. <https://doc.omnetpp.org/inet/api-current/neddoc/inet.applications.udpapp.UdpBasicBurst.html>

[11] J. Burkardt. “UNIFORM: A Uniform Random Number Generator (RNG).” John Burkardt. <https://people.math.sc.edu/Burkardt/cpp_src/uniform/uniform.html> (Accessed 6 August 2021).

[12] “4.12 Direct Method Calls Between Modules,” *OMNET++ Simulation Manual, OMNET++ version 5.6.1*. <https://doc.omnetpp.org/omnetpp/manual>.