An Experimental Application of Predictive Energy-Aware Routing (PEAR) in ESP Multi-Hop Mesh Networks

Alex Milton Sylvest Højlund Nielsen

Nikolaj Robert Fuglsang Andersen

Laust Nørskov Thygesen

Supervised by Maja Hanne Kirkeby

ABSTRACT

Abstract

Keywords:

Project source: https://github.com/NikolajRFA/painlessPEARMesh

CONTENTS

1 INTRODUCTION

The application of Internet of Things (IoT) is becoming increasingly widespread and enables extraction of data facilitated by machine-to-machine (M2M) communication. Depending on the area of application the communication is affected by various requirements. An application of IoT might have unlimited energy resources if connected to an outlet – but in certain scenarios these systems will be limited to an exhaustible power source such as batteries. In these scenarios certain precautions must be taken to extend the lifetime of the systems and thereby increase efficiency and reliability. Approaches to extend the lifetime of such systems using energy-aware routing of communication in industrial settings have been proposed. An example of a proposed approach is the energy-aware routing algorithm described by Jecan et al. (2022). An algorithm that balances the energy consumed by the nodes in a network in order to extend the lifetime of the network.

The PEAR algorithm was originally developed for the centralised ISA100.11a network, where a central unit controls the network's topology in order to achieve energy balancing and extended network lifetime. In this project, we investigate how PEAR's principles can be applied in a different network context – namely a painlessMesh network, which is inherently decentralised and dynamic. PainlessMesh is an open-source library used to manage meshes consisting of ESP8266 and ESP32 devices. "Dynamic" refers to something occurring in real-time (and not a precomputed optimal topology).

To make PEAR applicable, we have adapted the algorithm so that a central component still calculates and adjusts the network topology – understood as the connections between the nodes – while the rest of the network communication takes place in a distributed manner.

Instead of providing a theoretical proof of correctness, we have carried out an experimental evaluation based on the method used in the original PEAR article. In doing so, we assess how our adaptation affects the network's energy efficiency under comparable conditions.

This leads us to the objective of this project – as we seek to answer the question if the PEAR algorithm can be applied to non ISA100.11a compliant systems with similar benefits.

2 USE OF GENERATIVE AI TOOLS

In the preparation of this thesis, generative artificial intelligence (AI) tools such as OpenAI's ChatGPT were used to support the writing process. These tools assisted in refining the structure of the text, improving grammar and clarity, and suggesting phrasing for technical content based on our original ideas and input.

All core research contributions, technical implementations, experiments, analyses, and conclusions presented in this thesis are our own. The AI tools were used strictly as writing aids.

The use of AI-assisted tools was conducted responsibly and in accordance with the guidelines provided by Roskilde University.

2.1 Problem and Motivation

2.2 Definitions

This project specifically investigates the application of the PEAR algorithm within networks consisting of ESP8266 and ESP32 devices.

The inspected networks have the following characteristics:

- **Battery powered**: the devices are off-the-grid. Therefore, it is of utmost importance that the energy consumption is limited to extend the lifespan of the network.
- **Communication**: Communication within the network is handled by 'hopping' or 'bouncing' messages from node to node to reach the desired receiver. The object of the PEAR algorithm is to optimize the routes used for the bounces thus balancing the energy consumption throughout the network.

3 PROBLEM DEFINITION

The problem definition is as follows:

Is it possible to replicate the benefits created by PEAR in a non ISA100.11a compliant painlessmesh powered wireless multi--hop network, thereby enhancing the longevity and reliability of the network?

3.1 Research questions

To answer this problem definition we need to answer the following questions:

- How do we define a depleted or 'dead' network? In order to create a valid comparison between the test setups we need a set of rules that define the lifespan of a network.
- In which ways do we have to change the PEAR algorithm for it to fit the painlessMesh powered mesh?
- What input does the PEAR algorithm take that we need to extract from our nodes?
- How do we calculate the predicted lifespan of the nodes?
- Does the number of nodes in the network change the results of the experiments?

4 THEORY

This section will describe the foundation on which the project builds. First, PEAR will be described. Then painlessmesh, the area where the logic will be applied, is presented. Followed by a description of the experiments that will investigate the validity of the approach.

4.1 PEAR: Predictive energy-aware routing

This section describes PEAR. Specifically, what the objective of the algorithm is, and how the objective is achieved. This includes a description of the systems wherein PEAR is applied, with focus on the standards the systems comply

4.1.1 PEAR's objective

PEAR is an algorithm designed to be used in industrial wireless sensor networks complying with the ISA100.11a standard. The algorithm's objective is to predict and extend the lifetime of IWSN in order to ensure *"continuous system availability, cost efficiency and suitability for safety applications"* (Jecan et al., 2022). This is because *"In real use case scenarios, to ensure the economic and operational efficiency, battery-powered IWSN field devices require scheduled maintenance procedures, such as battery replacement for a subset of field devices having the same lifetime expectancy. IWSN maintenance procedures involve variable costs directly related to field device service and fixed costs associated with factory logistics and downtime. To minimize the fixed costs, the IWSN maintenance cycles should be predictable on a field device cluster basis"* (Jecan et al., 2022, p. 6).

This means that the algorithm configures the network in order to extend the lifespan of the network and makes sure devices are depleted equally within energy profile clusters. This is possible as PEAR monitors the network's devices and their energy margins, in order to utilise energy more evenly across the network. *"These margins create an energy potential in the network that is discovered and leveraged by PEAR."* (Jecan et al., 2022). To put this into a different perspective, a shortest path approach to routing communication will in most cases result in fewer messages across the entirety of the network, but could in some cases cause certain nodes to deplete more rapidly than others - resulting in decreased network lifetime. (Jecan et al., 2022) This would result in batteries having to be replaced individually instead of changing batteries for all devices in a given cluster at once.

4.1.2 Energy Profiles

The results presented by Jecan et al. (2022) include a figure describing network lifetime expectancy - specifically how much longer the networks where PEAR is applied are expected to be online. In order to create this figure *'energy profiles'* are used.

"[...] energy profiles are defined as the maximum energy consumption rate required to meet the lifetime expectation of field devices. From the PEAR algorithm perspective, the energy consumption rate is equivalated to the number of datalink protocol data units (DPDUs) transmission (Tx) and reception (Rx) opportunities per minute." (Jecan et al., 2022, p. 7)

In other words, Tx and Rx can be understood as transmitted and received messages, and the number of these messages determines how much energy a device consumes while sending and routing messages in the network. The energy profiles can be viewed as a restriction that a node has to comply with, in order to meet a desired lifetime.

4.1.3 Algorithm input: ISA100.11a, CDS VR950 IIoT Gateway hardware platform, and CDS VS210 field devices

As stated previously PEAR is designed for ISA100.11a compliant systems - and utilises the associated data and opportunities that are exposed as a result of the compliance with the standard. The algorithm relies on the CDS VR950 IIoT gateway that acts as a manager for CDS VS210 field devices (sensor devices that report data to the manager) . The VR950 gateway acts as an orchestrator: gathering data, running the PEAR algorithm and distributing the generated routing updates (Jecan et al., 2022). The flow of communication within

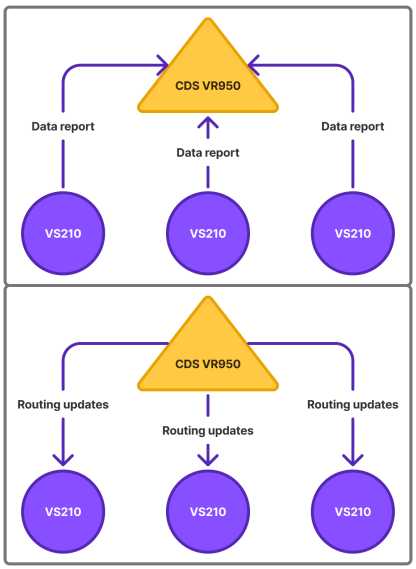

Figure 1. Communication flow in single level IWSN topology described by Jecan et al. (2022)

```
while The ISA100.11a management modules are running
 evaluate the routing graph every minute do
   for each device in the listOfAllDevices if
    noOfVerifiedDevices < 10 do
       if Function isEnergyNotInLimits(device)
          if noOfUsedTxSlots > txProfileThreshold and
           noOfUsedRxSlots > rxProfileThreshold then
             return False
          else
             noOfVerifiedDevices += 1
       then
          Function correctEnergyDevice(device)
             Create descending Rx rate childRxList;
             for each device in childRxList do
                if if backup parent is in energy limits
                 then
                    primaryParent = backupParent;
                    return True
                for each candidate in candidateList
                 do
                    if cadidate energy is in limits and
                     candidate is from higher profile
                     then
                        newParent = candidate;
                        return True
                    else
                        newParent = candidate;
                        break;
```

Figure 2. PEAR post-network-formation algorithm pseudocode (Jecan et al., 2022, p. 9)

the network is illustrated in Figure 1. The data reported by the VS210 field devices includes readings from *"[...] temperature, humidity and pressure sensors whose values are acquired and transmitted every 30 s within a single data packet."* Jecan et al. (2022). But these data reports also importantly include the number of receiving and transmission timeslots used by the device. These receiving and transmission timeslots are measured against the devices' energy profiles and based on this decisions are made towards creating a network where the communication is divided more equally.

Alongside these sensor readings the VS210 devices report *"[...] the quality of wireless links from a device perspective regarding their immediate neighbours."* (Jecan et al., 2022). The latter is garnered by the VR950 gateway and used when calculating possible routing optimisations. The communication from field device to VR950 gateway is facilitated by graph routing. The VR950 gateway allocates links, which are directed connections, for all field devices in the network. These links are appointed as preferred or backup links and depending on the availability of allocated links the field devices choose the preferred possibility for reporting data to the VR950 gateway (Jecan et al., 2022).

4.1.4 The algorithm

Routing algorithms differ from each other by employing different feature prioritisations - some algorithms produce routing that prioritise the lowest possible number of messages sent within the network, where other prioritise balancing the messages across the network in order to extend the lifetime of the network. The PEAR algorithm aims to increase the network lifetime, and as a by-product the total number of messages in the network is lowered. The algorithm does this by iteratively restructuring the routing of communication throughout the network. The reason why the rerouting is done iteratively, specifically on a maximum of ten devices per run of the algorithm, is to ensure the processing load remains low enough to not cause interference with the network stability (Jecan et al., 2022). This constraint means the VR950 manager will have to run the algorithm a number of times to correct the energy expenditure of all devices. Furthermore, if a device's communication is rerouted, then intuitively all upstream connections from the device will have to be re-evaluated, as the rerouted communication could have pushed other devices over their message threshold.

The algorithm, as described in the pseudocode displayed on Figure 2, is structured as a loop through a list of all devices in the network, which continues until the aforementioned ten devices (at most) are evaluated. For each device the energy usage is compared with the device's energy threshold, and if found exceeding the algorithm will attempt to correct the device's energy expenditure - else the number of verified devices is incremented. When attempting to correct a device's energy expenditure, a list of downstream connections or children is created - sorted descendingly by the child's number of transmitted messages.[1] A child's number of transmitted messages describes the load that the child puts on the child's parent node. The algorithm runs through this list of children and attempts to reroute the children with the highest energy consumption to a new parent candidate. First attempt to reroute a child is to view the child's backup parent, which is a backup communication link allocated by the VR950 manager. If the backup exceeds its own energy limit or belongs to an equal or lower energy profile, then a list of the child's other parent candidates is made and looped through. The new parent candidate's energy consumption and energy profile is checked to verify if the parent candidate is a viable candidate. The candidate has to be within its energy limit and from a higher energy profile - meaning the parent candidate is allowed to spent more energy compared to the child being rerouted while respecting the parent candidate's energy profile. This flow is visualised on Figure 3. This causes a recurring reconfiguration of the network's communication, where devices are continually checked and rerouted. However, the reconfiguration is constrained by the requirements of the ISA100.11.a standard:

> *"The PEAR algorithm aims to provide energy balancing support without interfering with the control and monitoring performance. The evaluation period was empirically set to one minute, which is considered large enough not to affect the efficiency and performance of the ISA100.11a management modules and small enough to provide more routing change opportunities for the PEAR algorithm." (Jecan et al., 2022, p. 8).*

[1] The PEAR pseudocode states the children are sorted by Rx descendingly and not Tx, however the paragraph above states *"[...] the procedure creates a list with all the children of the overconsuming device arranged in descending order of the Tx rate. The Tx rate indicates the load that a field device adds to the parent."* (Jecan et al., 2022, p. 9). Therefore, we have concluded that the pseudocode is meant to state Tx and this is the logic we have implemented in our updated implementation of the algorithm.

To reiterate, the evaluation period—the time between PEAR runs—is long enough to avoid destabilizing the network's other tasks, yet short enough to allow timely reconfiguration. For example, if the network required significant reconfiguration but only ran once per hour, achieving stability could take an excessively long time.

The end state of the algorithm, described as PEAR stability by Jecan et al. (2022), is when *"[...] all possible energy balancing topology changes have been exhausted"* (Jecan et al., 2022, p. 12). This means that either all field devices fully comply with their respective energy profiles, or some devices failed to comply but all possible optimisations have been attempted.

4.1.5 PEAR experiments

The experiments carried out by Jecan et al. (2022) include test setups with what can be described as homogenous and heterogenous networks in terms of energy profiles. Both test setups are compared with a setup of ISA100.11a network that does not run the PEAR algorithm. In the following, we will describe the tests conducted by Jecan et al. (2022) and the results gathered by those.

Homogenous Jecan et al. (2022) make homogenous tests of the PEAR algorithm implemented in the ISA100.11a with eight different energy profiles. Intuitively, Jecan et al. (2022) achieve an increasingly higher PEAR success rate when the energy profile is raised. Interestingly, (Jecan et al., 2022, p. 12-13) find that increasing the energy profile from $200\frac{DPDU}{min}$ to $250\frac{DPDU}{min}$ only gives give a 10% increase to the success rate $(100 - \frac{numberOfOverconsumingNodes}{numberOfNodes} * 100\%)$ of the PEAR network, from 90% to 92.85%. This trend continues when raising the energy profile further from $250\frac{DPDU}{min}$ through $450\frac{DPDU}{min}$. According to (Jecan et al., 2022, p. 12) this behaviour is indicative of PEAR not being able to find a better topology while retaining network stability. It is worth to note, that the opposite is apparent when increasing the energy profile from $100\frac{DPDU}{min}$ through $200\frac{DPDU}{min}$ where we see an increase from 48.57% to 90% in the success rate (Jecan et al., 2022, p. 13). In Figure 9 of Jecan et al. (2022) it is evident that over time and with topology changes, the average energy consumption by a field device is reduced. The figure shows temporary energy inefficiencies before larger drops in average energy consumption.

> *"When trying to lower the energy consumption*

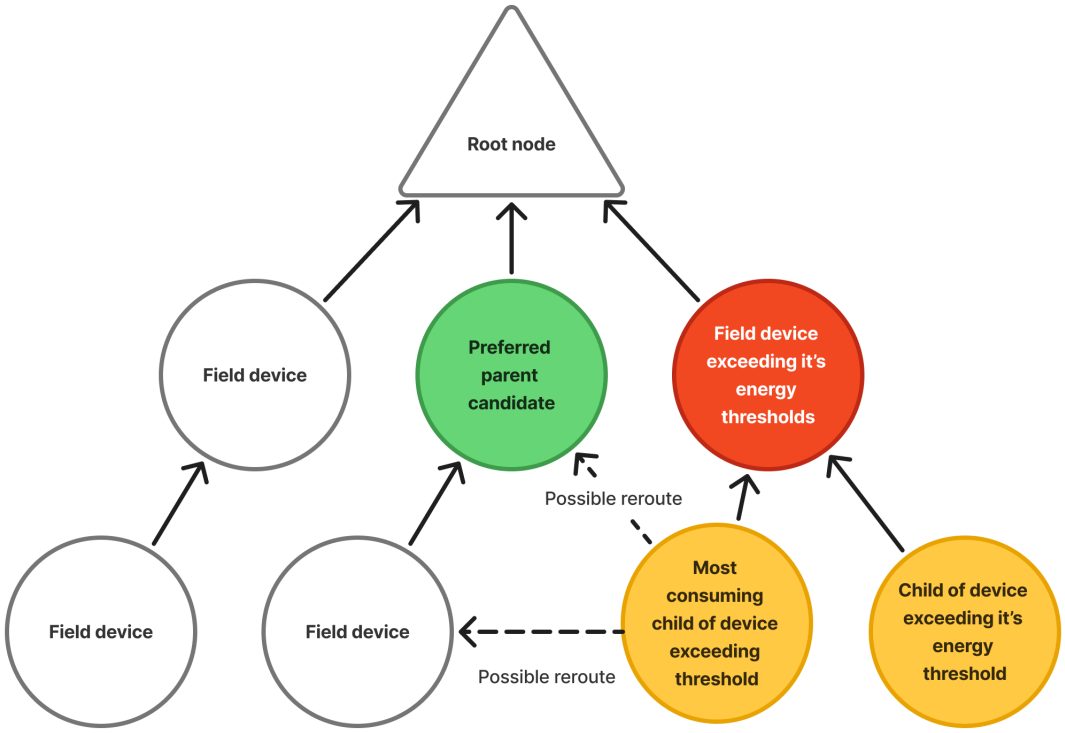

Figure 3. Correcting a device's energy consumption by rerouting child

> *of a field device, the PEAR choices are limited by the set neighbours of each device. This leads to cases where an overconsuming field device cannot be freed from its child nodes due to limited choices. Therefore, PEAR triggers intermediary network topologies that create new opportunities to free the overconsuming field devices.*
> *(Jecan et al., 2022, p. 14)*

The energy inefficiencies apparent before a drop in average energy consumption can therefore be attributed to the PEAR algorithm reconfiguring the network in order to make new opportunities for further optimising the network later. Jecan et al. (2022) sum up the performance of PEAR in networks with homogenous energy profiles as reducing overconsumption with 10.4 times while increasing the average field device energy consumption with 1.8%. This increase in energy consumption compared to the network without PEAR emphasizes that the objective of PEAR is not to directly reduce the energy consumption of the network, but rather to prioritise the predictability of energy consumption and energy balance (Jecan et al., 2022, p. 14).

Heterogenous Jecan et al. (2022) also evaluated the PEAR algorithm on networks where the nodes had different energy profiles. For these tests the 70 field devices are split into two clusters with different energy profiles. The clusters are based on expected node lifetime, leaving the configuration with a cluster where the expected lifetime is short, and energy consumption thus is high, and one where the lifetime is long, and energy consumption thus is low. The authors proceed test seven different splits of the 70 field devices, varying from 50 having the high energy consumption energy profile and 20 having the low energy consumption energy profile, to five

having the high profile and 65 having the low profile. The objective of PEAR in this configuration is having the field devices respect their energy profiles, thus providing energy consumption and field device lifetime predictability (Jecan et al., 2022, p. 17). In figure 10 and 11 the authors show how, with 20 field devices in the high consuming energy profile and 50 in the low consuming energy profile (H20L50), the PEAR algorithm can balance the topology so all field devices are within their energy profile. Before the PEAR algorithm is run on the network 18.5% of field devices in the network is over consuming, however, the PEAR algorithm is able to lower this to 0% through topology changes (Jecan et al., 2022, p. 16-17). Another discovery in the H20L50 test is that the average energy consumption for the high energy consumption energy profile cluster is lowered by 25% and the average energy consumption for the low energy consumption energy profile cluster is lowered by 23.6%. This shows for this test with multiple energy profiles defined, the overall energy consumption of the network is lowered (Jecan et al., 2022, p. 17). This is in contrast to the 1.8% increase in energy consumption that was reported for the homogenous energy profile tests. For this test, PEAR achieved a 100% success rate across both energy profile clusters, up from 84% on the low energy consumption energy profile cluster, and 75% on the high energy consumption energy profile cluster (Jecan et al., 2022, p. 17). The performance statistics for different splits of the 70 field devices into high and low energy consumption energy profiles, show that when there are more field devices in the high energy consumption energy profile cluster, it is easier for the PEAR algorithm to balance the network and therefore reach the expected lifetimes for the field devices. Regardless of the actual split between the field devices, PEAR offers a 2.31 times better cluster lifetime than a network without PEAR on average (Jecan et al., 2022, p. 18).

To conclude on their tests, (Jecan et al., 2022, p. 20) state:

> *"if the energy margin (defined as the difference between the energy profile and the average field device consumption) is at least 50 data packets/minute, the PEAR solution can ensure the expected cluster lifetime. Moreover, PEAR extends the cluster lifetime by up to 3.2 times while reducing the average energy consumption by 23.6% and the total overconsumption 10.4 times after triggering 210 network topology changes."*

Therefore, it is conclusive that PEAR has a positive impact on the predictability of the energy consumption of field devices in a ISA100.11a based mesh network.

4.2 PainlessMesh

PainlessMesh is a library that is being implemented into the project as it facilitates the creation of a WiFi mesh network for our ESP devices. Whilst PEAR will be used to optimize our network, the painlessMesh library will be creating the network itself. Whilst the name might suggest that the topology of a painlessMesh network is a mesh, the topology is more reminiscent of a star network though with a few notable changes that makes it differ from ordinary star networks. One of these changes is removing the singular access point that most star topology networks are made up of, in a painlessMesh network multiple access points can exist Yoppy et al. (2019). This allows a painlessMesh network to facilitate mesh communication, as each node can be used as an access point means that a node is capable of both receiving and transmitting messages, whilst the painlessMesh network does not offer cyclic paths due to resource constraints Yoppy et al. (2019). PainlessMesh does allow the network to send messages between nodes, and not only to a singular access point. This allows nodes that would not have a connection to a potential root node, to be able to still send data towards the root node, with the help of neighbour nodes acting as an access point.

4.3 PainlessMesh changes

PainlessMesh as a base, is incapable of supporting PEAR. We had to make a series of changes to be able to make sure that PEAR and painlessMesh could work together. An unaltered painlessMesh node will simply choose which neighbouring node it will connect to based on RSSI. If a signal to one node is better than another, it will choose the strongest connection and form a connection to that node. Once that node has formed a connection, if a new node is then introduced into the network, it increases the chances of painlessMesh causing a reconfiguration . If any of the other nodes in the network prefers the new node's RSSI value, during the reconfiguration, they will be conneceted to the new node. This will cause issues when PEAR needs to interact with painlessMesh. Nodes are sending PEAR data towards the root node, which will decide based on messages transmitted and received, if a node should be rerouted to a different node. If the node, due to painlessMesh already has been rerouted, it will ren-

der the data and potential reroute from PEAR useless. As PEAR is stating to a node that it needs to reroute with data that would no longer be applicable, due to the changes made by painlessMesh. We needed to find a way to make sure that when a node made a connection with another node, that the connection would not change unless otherwise told by the PEAR algorithm, and not randomly with painlessMesh's automatic reconfiguration. To achieve this goal we implemented *targetNodeId*. During startup of the network, a network scan is initiated. Meaning that every node with the same SSID will be incorporated into the mesh, once a node connects to the mesh, the *targetNodeId* when first used during a bootup sequence is still defined by RSSI. However, once the strongest RSSI has been found, the node will be given a *targetNodeId*. Once the node has identified the target node, it will connect to the node matching the *targetnodeId*, and remain connected to the given *targetNodeId* until that value is later updated due to the PEAR algorithm. This secures our connections will remain stable between PEAR runs, as well as ensuring that nodes will not randomly reconfigure due to painlessMesh.

To support the implementation of targetNodeId. We have constructed two functions to help us retain and make connections, namely *compareWiFiAPRecords()* and *checkStation()*. As described earlier, we wanted to be able to facilitate PEAR within the painlessMesh network without disrupting painlessMesh's normal flow. We have enhanced, and tweaked how painlessMesh works to make room for PEAR optimization. *compareWiFiAPRecords* compares potential access point connections. A connection is chosen based on the *targetNodeId*, if the compared access point connections contains a *targetNodeId* a connection will be made to the *targetNodeId*. If the compared connections do not contain a *targetNodeId*, the connection will instead be made with the access point that has the best RSSI value. *checkStation()*, makes sure that a node is connected to their *targetNodeId*. When the *checkStation()* function is called, it will check a node's station, and check if it is their *targetNodeId*, if it is. It will remain connected to that station. If it is not, the station will make a new *targetRecord*, which will be used to update the connected node's connection so that on the next *stationScan*, it will be redirected to the *targetNodeId*. The *targetNodeId* secures that the network is only rerouted due to PEAR, and that if a reconfiguration from painlessMesh does occur, that the nodes that would be affected by a reconfiguration, remains connected to the correct nodes, until they are otherwise told to be rerouted by a PEAR run.

We encountered a different problem further into the development process. PainlessMesh uses the word *subs* to describe connections. The word *subs* would normally indicate that we are only looking at a node's children, in painlessMesh this is not the case. Instead, *subs* looks at all connections, not just downward connections. This means that a node's parent, is also in the *subs* list. If we don't remove the parent from the list, PEAR will be able to tell a child node that it needs to reroute its parent, which can quickly cause network instability. An example is that a node that is connected directly to the root node, would be told by PEAR that it needs to be rerouted to one if it's child nodes. The nodes that are directly connected to the root node are coded to always be rerouted back to the root node, so the node is now connected to one of it's children, and trying to reconnect back to the root. Which causes the node's other children to reroute as well. During experiments, we encountered that the node which should be connected to the root, would never be reconnected with the root. To combat this issue we added a function called *removeStationFromAvailableNetworksIfInNodeSubs()*, once a *stationScan* is initiated. As we know what station, or parent a node is connected to. We can now remove the parent network if the parent node exists within the *subs* list of a child. *removeStationFromAvailableNetworksIfInNodeSubs()* will help with network stability, and make sure that the nodes which should be connected directly to the root, will always prefer and stay conneceted to the root node.

5 DESIGN

This section describes our approach to applying the PEAR algorithm on a painlessMesh network.

5.1 Overview of algorithm input

To start the development process we analysed the PEAR pseudocode (Figure 2) in order to understand the data that we needed to expose in the painlessMesh network. The data needed to run the PEAR algorithm is as follows:

- A list of all devices in the network
- The number of Tx and Rx slots used by nodes
- The nodes' Tx and Rx thresholds
- A designated backup parent for all nodes

- A list of children (immediate downstream connections) for all nodes
- A list of parent candidates (visible but non-connected nodes) for all nodes

5.1.1 List of all devices in the network

To be able to check the energy consumption of all nodes in the system a list of all nodes is needed. Jecan et al. (2022) describes that the PEAR algorithm works as a loop over a *listOfAllDevices* but do not describe how this list is generated. Therefore, it was decided that the list would be generated as a breadth first search of the tree from the root nodes perspective. Meaning, nodes closer to the root would be positioned first in the list.

5.1.2 Data report from nodes

The field devices in the network investigated by Jecan et al. (2022) report sensor readings and Tx/Rx timeslots to the manager every 30 seconds. A similar approach was decided for in this project where nodes on a one minute interval report

- the number of received and sent messages,
- and available but not yet connected nodes

to the root node. This data is used in combination with the data already exposed by painlessMesh, namely the structure of the mesh including upstream and downstream connections.

5.1.3 Backup parent

The VR950 manager allocates backup parents for nodes in the network investigated by Jecan et al. (2022) which differentiates the IWSN from the painlessMesh network. We found that as painlessMesh does not hold similar redundant connections we would simply skip this step of the algorithm, and treat all parent candidates the same. As we do not know the specifics of how a backup parent is defined, other than it intuitively being a backup, it is difficult to assess what impact this has on the updated algorithm and the generated results.

5.2 Facilitating PEAR in painlessMesh

In order to facilitate running the PEAR algorithm in a painlessMesh network various prerequisites arise. This section describes how we have updated painlessMesh to meet these requirements.

5.2.1 Data reports

To start off, the field devices need a way to report data to the root node, and the root node needs a way to distinguish which data is PEAR related data. The painlessMesh API exposes the *sendSingle* method (Martín et al., 2024) which is a method that takes a message and forwards it to a specific destination. The message is wrapped in a *Variant* object, and it was decided to extend the variant class to hold a PEAR variant, to enable nodes in marking messages as containing PEAR data. On the receiving end, the root node, an on receive callback (*onPearReceive*) is configured to process the received PEAR data. The root node takes this data and updates the representation of the nodes, thus enabling the root node in running the PEAR algorithm on a set of data that reflects the current configuration of the network. If a PEAR run results in generated reroutes, the reroutes will be send to the field devices using the same PEAR message variant. The flow can be viewed on Figure 5. The decision to implement a stand-alone PEAR message flow is based on the goal of keeping PEAR functionality separated from the ordinary painlessMesh functionality. The developed PEAR functionality can be viewed as a behind the scenes routing optimisation that we wanted to be abstracted away from the application layer in order to maintain the normal painlessMesh functionality as is.

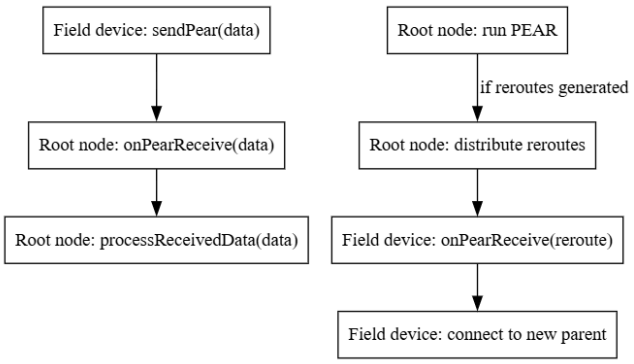

Figure 5. Description of the flow of PEAR messages in the network

5.2.2 Bypassing the algorithm

As described above, the field devices report data to the root node, which triggers the *onPearReceive* logic. This logic processes the received PEAR data to be used in the PEAR algorithm. However, we chose to add logic to the data processing step that enabled removal of redundant and potentially negative impacting runs of the PEAR algorithm. This is done

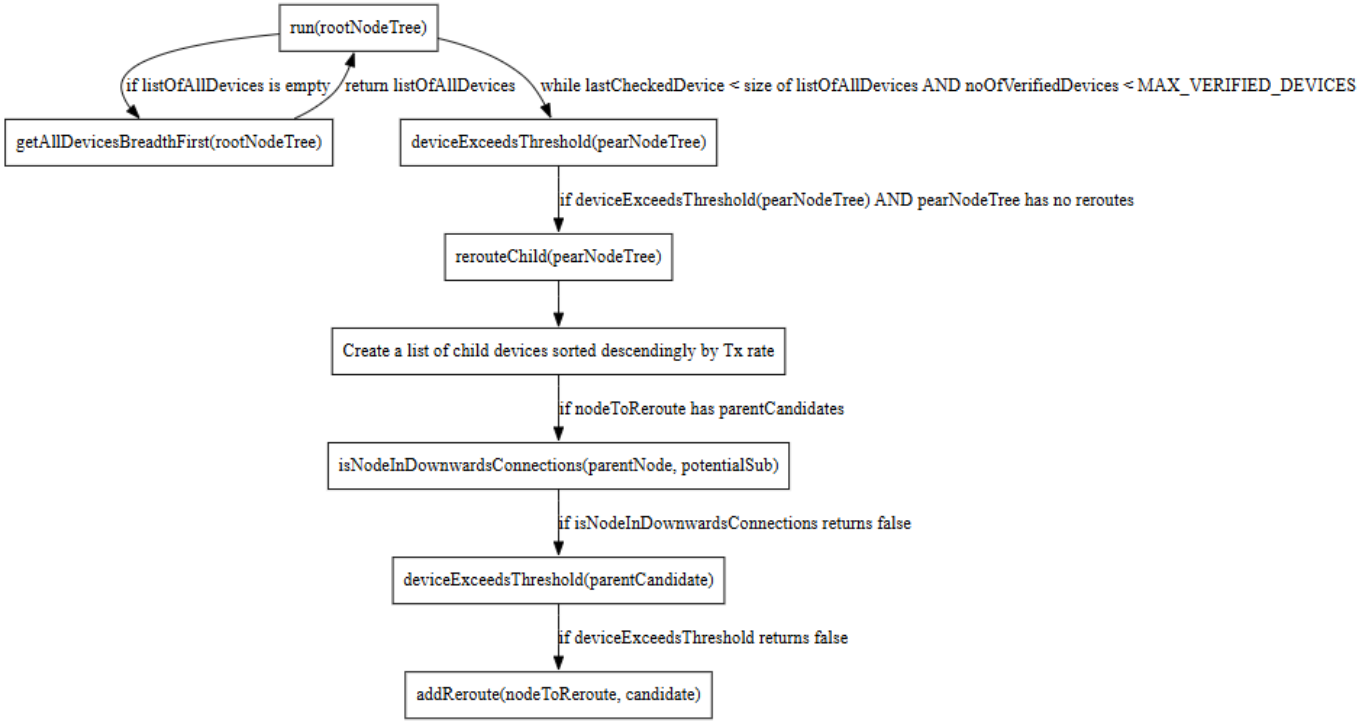

Figure 4. The flow of the updated algorithm

by analysing the received field device's parent candidates, and check whether the root node is present in the list. If so, a reroute is scheduled immediately, without further computation. Now, as part of the actual run of the PEAR algorithm, the scheduled reroutes is checked, and if the node being processed by the algorithm already has a reroute it is simply skipped. Intuitively, field devices should always be rerouted to the root node if possible, as all data reports are targeting the root node. We have found that skipping nodes with already scheduled reroutes is incorrect behaviour - just because a reroute to the root is scheduled, the node should still be energy corrected if needed. However, as the algorithm verifies devices multiple times, the skipped devices are checked in later runs.

5.2.3 PainlessMesh tree configuration

PainlessMesh in its normal configuration works as a decentralised mesh where all nodes are equal in their ability to connect to other nodes, i.e. all nodes can create an outgoing connection and hold multiple incoming connections. In this configuration, the root node - as all other nodes - attempts to create an outgoing connection. This creates tree structures where the root node is able to be positioned at any given position of the tree, which does not comply with the PEAR algorithm's logic. PEAR is originally applied within centralised networks where the manager is positioned as the root of the tree. Therefore, we have chosen to remove the root node's opportunity to create outgoing connections.

6 EXPERIMENT DESIGN

We conducted a series of experiments and tests to evaluate whether our implementation of PEAR delivered the expected outcomes. Specifically, we aimed to determine if PEAR reduced the number of over-consuming devices. Additionally, we observed the algorithm's success rate by measuring how many nodes are within their energy profiles when the experiments are concluded. The "Test Setup and Experiment Design" chapter details our experimental setup, the key variables we monitored, and the results from multiple runs of the PEAR implementation.

6.1 Test setup / experiment design

6.1.1 Boot Sequence

The root node is initialized first. As it cannot initiate STA (Station) or outgoing connections on its own, a secondary node capable of establishing a connection with the root is subsequently booted. This ensures the root is integrated into the network. Following this, the remaining nodes are booted in an

order determined by their respective energy profiles—nodes with higher energy availability are prioritized and initialized earlier. Upon booting the root node, a timer is initiated. Once this timer reaches five minutes, the PEAR (Power Efficient Adaptive Routing) algorithm is executed for the first time. This operation is scheduled through the TaskScheduler using the enableDelayed method on the runPearTask. Upon the initial execution of the PEAR algorithm, a secondary timer is initiated to measure the duration until the network reaches a stable state.

Upon the initial execution of the PEAR algorithm, a secondary timer is initiated to measure the duration until the network reaches a stable state.

Network stability is defined as the condition where all nodes operate in accordance with their energy profiles. Stability is assessed by repeatedly executing the PEAR algorithm until no further node rerouting occurs across a defined number of consecutive runs.

The number of rerun iterations required to confirm stability is computed as:

$$\frac{N+M-1}{M} \tag{1}$$

where:

- N is the total number of devices, *listOfAllDevices.size()*,
- M is the value of *MAX_VERIFIED_DEVICES*.

This calculation ensures that each device is verified at least once during the non-rerouting phase.

Example: Given a network with $N = 45$ nodes and $M = 10$, we compute:

$$\frac{45+10-1}{10} = \frac{54}{10} = 5$$

Thus, the network is considered stable after **5** consecutive runs with no reroutes in the given example.

6.1.2 Experiment configurations

We have outlined different experiment setups to cover the configurations and testing mentioned in the PEAR article. The experiments consists of setups with:

- Heterogenous networks with multiple energy profiles across nodes,
- Homogenous networks with a single energy profile across all nodes,
- and finally a control setup where PEAR is disabled.

The composition of these experiments is aimed at depicting PEAR's capabilities of extending the network lifetime across networks consisting of different energy profile configurations. The first experiment conducted was the 'PEAR-disabled' experiment, which served as a baseline. The energy profiles for subsequent experiments were selected based on insights gained from this initial test. We inspected the amount of packets sent by the nodes, and created an energy profile with a lenient energy threshold. With this lenient energy profile defined we would go on to sequentially decrease the threshold until the compliance with the energy profiles would decrease significantly.

The experiments will include independent, dependent, and controlled variables, defined as follows. According to Wohlin et al. (2024, p. 77):

> *"Those variables that we want to study to see the effect of the changes in the independent variables are called dependent variables (or response variables). Often, there is only one dependent variable in an experiment. All variables in a process that are manipulated and controlled are called independent variables."*

From this, we understand that independent variables are those we deliberately manipulate during the experiment. Dependent variables are those we observe and measure to assess the impact of changes in the independent variables, and they are used to support or refute our hypothesis. Controlled variables remain constant throughout the experiment to prevent them from influencing the results; for example, the boot-up sequence will be held constant in all trials.

To set up our experiments, we identified the independent, dependent, and controlled variables, as well as the hypotheses. These experiments tests the **hypothesis**:

- The use of PEAR reduces the number of over-consuming devices compared to a similar network where PEAR is disabled.

Corresponding to the following **null hypotheses**:

- PEAR has no effect on the number of over-consuming devices.

The **independent variable** in these experiments is the *energy profile*, as this is the variable we manipulate across different runs. The **dependent variables** are (1) the number of over-consuming devices, and (2) the success rate. The success rate is defined as the percentage of devices that remain within their energy profiles after the network has reached a stable state.

The **controlled variables** are the aspects of the experiment that remain constant throughout all runs. These include:

- **Number of nodes:** Always set to 20.
- **Node firmware:** Remains the same for all nodes, except for the configured energy profile.
- **Boot-up sequence:** Consistently follows the same order in every run.

The boot-up sequence begins with nodes assigned energy profile 1, followed by nodes with profiles 2, 3, and finally 4. Nodes are added to the network one at a time until all 20 are connected. This sequence follows the experimental setup described in the PEAR article, where high-energy-profile nodes are connected before lower-energy-profile ones. For experiments using only two energy profiles, we follow the same boot-up pattern as in the PEAR study.

Control experiments are run with PEAR disabled. These experiments follow the same boot-up sequence as the PEAR-enabled tests. However, because a painlessMesh network without PEAR cannot achieve a stable state, control experiments are allowed to run for six minutes after all 20 nodes have connected.

The experiments are done on 20 nodes, following the boot sequence outlined earlier. Each log is marked with its date, and the energy profiles used for each of the nodes. Following these experiments, we will be plotting our data into graphs and tables to better understand our results and see if we can prove or reject our hypotheses.

6.1.3 Experiment challenges

Due to the nature of how time synchronization works for our messages, we would sometimes run into nodes sending more data than usual, though we see this as a benefit, as in a real world scenario extra data such as a warning on battery level, or drastic changes in field data would warrant more data being sent every now and again. An experiment run will be considered complete after the network has been declared stable, and has run PEAR at least once after being stable to see if any reroutes might still occur due to time synchronization messages.

Homogenous vs. heterogenous	Pear or No-pear	Energy profile (Rx/Tx)
N/A	NoPEAR	N/A
Heterogenous	PEAR	1. 55tx/50rx 2. 50tx/45rx 3. 45tx/40rx 4. 40tx/35rx
Heterogenous	PEAR	1. 50tx/45rx 2. 45tx/40rx 3. 40tx/35rx 4. 35tx/30rx
Heterogenous	PEAR	1. 45tx/40rx 2. 40tx/35rx 3. 35tx/30rx 4. 30tx/25rx
Heterogenous	PEAR	1. 40tx/35rx 2. 35tx/30rx 3. 30tx/25rx 4. 25tx/20rx
Heterogenous	PEAR	1. 35tx/30rx 2. 30tx/25rx 3. 25tx/20rx 4. 20tx/15rx
Heterogenous	PEAR	1. 30tx/25rx 2. 25tx/20rx 3. 20tx/15rx 4. 15tx/10rx
Heterogenous	PEAR	1. 25tx/20rx 2. 20tx/15rx 3. 15tx/10rx 4. 10tx/5x
Heterogenous	PEAR	1. 65tx/45rx 2. 20tx/15rx
Homogenous	PEAR	All Nodes: 30tx/25rx
Homogenous	PEAR	All Nodes: 25tx/20rx
Homogenous	PEAR	All Nodes: 20tx/15rx

Table 1. Experiment configurations

7 VALIDATION

This section describes the derived results from the experiments.

7.1 Experiment data

Throughout our tests, we collected data in the form of logs from the root node of the mesh network. This meant that we had to tailor the logs the node reported to contain the data we were interested in analysing after our experiments. The data that is ultimately reported by the root node is a combination of data which is already used in PEAR, and supplemental data that we deemed needed in analysing the experiments. The data we ended up collecting is the data reported by each node as part of PEAR: *node_id*, *tx_period*, *rx_period*, and *parent_candidates*. To supplement this data, we added the *node_time* at which a PEAR report was made. This was done through the *getNodeTime()* method present in the *Mesh* class.

This node time value is synced between the nodes, this means that even though we have a boot sequence where network nodes are not all booted up at the same time, the *node_time* will be consistent across the nodes. Therefore, we can use the node time metric to keep track of exactly when a node has reported to the root node, even if the message is some time underway. We added a special logging level to painlessMesh, *DATA* which we used for logging the data we later wanted to extract from the logs. We then coded a log parser in Python that would look at these *DATA* logs, and extract the data within. The way the data was logged was in part as integers, as well as a json object with the PEAR report data. Our Python parser would then parse this data into a data class *PearReport* which we can then turn into a Pandas dataframe and further work on the analyse our experiment results. In the following section will we dive into the experiment results, and analysis thereof.

7.2 Experiment results

As Table 1 shows, our experiments can be divided into two groups. One where the set energy profiles of the nodes are homogenous throughout the network, and one where the set of nodes in the network is divided into groups where the energy profiles are homogenous, but the energy profiles are different between the subgroups of the network, we label these experiments as having heterogenous energy profiles in the network. Contrary to (Jecan et al., 2022), our mechanism for detecting PEAR stability is far simpler. Therefore, our data and analysis seeks to prove that PEAR reduces the amount of over-consuming nodes, and the amount which over-consuming nodes are over-consuming with. In order to analyse and gather results from the data from our experiments, we used Python to perform exploratory data analysis and create plots we used to further our understanding of the data from the experiments.

7.2.1 Homogenous network experiments

Our experiments include three experiments where all nodes have the same energy profile applied - i.e.: experiments depicting a homogenous network. The experiments can be found described in Table 1. To assess the performance of PEAR on homogenous networks we decided to make a baseline reading of how the meshing algorithm behaves without PEAR being active. This experiment is labelled NoPEAR in Table 1. This creates a baseline that we can compare the generated results to. For our experiments we have used a measurement of how many packets a node has transmitted as a proxy for the energy consumption of a node. This is because all packages a node receives are relayed towards the root node, therefore transmitted packages is always greater than received packages, as no packages are addressed directly to a node that is part of the network.

Table 2 shows the three tests we have made with one homogenous energy profile for the whole network. Similarly to Jecan et al. (2022) we wanted to test how different homogenous energy profiles would change the results of our PEAR runs. Therefore, we made three tests each with a slightly different energy profile. Table 2 assesses, for each experiment, the amount of nodes that were overconsuming before and after PEAR was run in the network. The data is based on whether the average number of packages a node has transmitted over the last two minutes is more or less than the energy profile threshold for transmissions. If a node transmits more packages than then transmit threshold of the energy profile allows, this node will be marked as an overconsuming node. From Table 2 it is evident that PEAR is able to reduce the amount of overconsuming nodes in the network, by balancing the package load between the nodes. Having overconsuming nodes after the PEAR run does not mean that running PEAR was not successful. Across all the tested energy profiles we have observed PEAR lowering the amount by which the node that is overconsuming the most is overconsuming. From our data we can see that the max overconsumption change on the experiment with Tx/Rx threshold 20-15 is -27.35%. The significance of this number is that eventhough there are still three nodes that are overconsuming after running the PEAR algorithm, the node that overconsumes the most, overconsumes 27.35% less than before PEAR was run. With the PEAR definition of when a network cluster dies being when the first node runs out of power, we can send $\sim 27\%$ more messages before running out of power than if PEAR was not run on the homogenous network.

Tx/Rx threshold	Pre-PEAR	Post-PEAR	Success rate	Max overconsumption change
30-25	2	1	94.74%	-14.75%
25-20	4	1	94.74%	-17.88%
20-15	7	3	84.21%	-27.35%

Table 2. Homogenous experiment overconsumption statistics

Figure 6 visualises these energy balancing optimisations that PEAR have made in the network, we have made a bar chart that shows the average package transmissions for each node over two minute periods, both before and after running

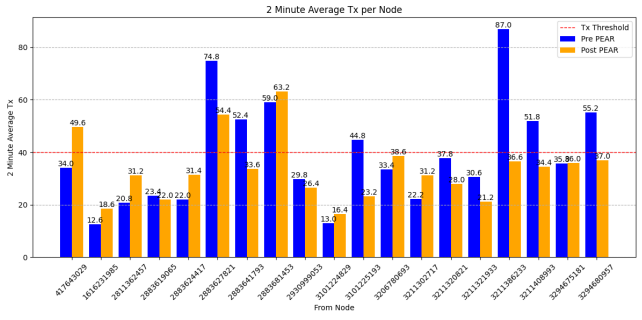

Figure 6. Tx/Rx threshold 20-15 bar chart showing average transmissions pre and post PEAR across 5 experiment runs

the PEAR algorithm. This bar chart visualises some of the same data that is also present on Table 2, but it also visualises cases where there is an energy potential between the pre-PEAR level of transmissions, and the transmission threshold. In cases where there is a gap between the pre-PEAR transmissions and the transmission threshold, the PEAR algorithm can reroute nodes in a way that utilises this energy potential. This highlights the prioritisation of nodes complying with the energy profile thresholds as opposed to lowering package transmissions across all nodes in the network.

From the figures displayed in Table 2 we can conclude that running the PEAR algorithm benefits the network. Our experiments have documented that the PEAR algorithm is able to balance package transmissions across a homogenous network with 19 nodes. Specifically with a transmission threshold of 20 and a received transmission threshold of 15 we have documented a 27.35% reduction of transmissions between the most over-consuming nodes, when comparing the transmissions before and after PEAR. We have also documented that the PEAR algorithm is able to balance the transmissions by shifting routes to utilise energy potentials present on under-utilised nodes. Thereby, creating a network where the difference in transmissions is lower.

8 DISCUSSION

Is it possible to compare our results with the results from the PEAR article? Yes/no? Howcome?

9 CONCLUSION

REFERENCES

Jecan, E., Pop, C., Ratiu, O., and Puschita, E. (2022). Predictive energy-aware routing solution for industrial iot evaluated on a wsn hardware platform. *Sensors*, 22(6).

Martín, G., BlackEdder, and contributors (2024). painlessmesh: A painless way to setup a mesh with esp8266 and esp32 devices. Accessed May 27, 2025.

Yoppy, Harry Arjadi, R., Setyaningsih, E., Wibowo, P., and Sudrajat, M. I. (2019). Performance evaluation of esp8266 mesh networks. *Journal of Physics: Conference Series*, 1230(1):012023.