An Energy Balanced Flooding Algorithm for a BLE Mesh Network

ADRIANO SOUZA BRANDÃO, MATHEUS CORDEIRO LIMA, CLAUDIA JACY BARENCO ABBAS, AND LUIS JAVIER GARCÍA VILLALBA

1 Faculdade de Tecnologia, Universidade de Brasília, Brasília 70297-400, Brazil
2 Group of Analysis, Security and Systems (GASS), Department of Software Engineering and Artificial Intelligence (DISIA), Faculty of Computer Science and Engineering, Office 431, Universidad Complutense de Madrid (UCM), 28040 Madrid, Spain

Corresponding author: Luis Javier García Villalba (javiergv@fdi.ucm.es)

ABSTRACT This paper presents a proposal for a balanced flooding protocol for mesh networks over BLE (Bluetooth Low Energy). The primary goal is to save battery lifetime by converting nodes into a mesh network in a balanced mode and thus increasing the lifetime of the network. Based on an extension of the flooding protocol Trickle, the proposed protocol, Drypp, aims to balance the loads of the network nodes. Hence it seeks to make nodes with a higher load more active in the network in comparison to nodes with a lower load. A performance analysis will be executed, comparing the proposed protocol and Trickle. The metrics of node lifetime, network lifetime, throughput and node battery loads will be analyzed.

INDEX TERMS Bluetooth low energy, bluetooth mesh, flooding, fruitymesh, low battery consumption, nordic, RedBear.

I. INTRODUCTION BLE operates in the 2.4 GHz industrial, scientific and medical (ISM) band and uses frequencies between 2402 and 2480 MHz. The spectrum used is divided into several channels, BLE uses channels of a 2 MHz spacing, which leads to 40 usable channels. These 40 channels are divided into three primary advertisement channels (37, 38 and 39), avoiding the main channels used by Wi-Fi, and 37 connection oriented channels [1]. This is shown in Figure 1.

The maximum data rate supported by last radio version is 2 Mbps. A typical range is 10-30 meters, depending on the environment surrounding the communicating BLE devices as well as the mode used. Bluetooth versions (when related to BLE) are backwards compatible with each other. However, the communication may be limited to the features of the older version of the two communicating devices [3].

The advertisement channels are used in broadcast mode. The Time Division Multiple Access (TDMA) scheme is used to synchronize bidirectional communication and each connection uses a Frequency-Hopping Spread Spectrum (FHSS) technique to ensure robustness.

When in advertising mode, to reveal its presence, a device periodically sends an advertisement package, alternating between the three channels reserved for this purpose. If a listening device wants to initiate a connection, it must transmit a request on the same channel on which it received the advertisement packet.

Once the connection is accepted, a piconet is formed and the device that requested the connection assumes the role of the master. Meanwhile the other device becomes the slave of the connection. There is the possibility of a device assuming, alternately and simultaneously, the roles of master and slave, being able to communicate with more than one master (in the role of slave) and/or being the master of several devices. Encrypted communication requires the devices to be paired.

The three main blocks of a BLE device architecture are: the application block, the host block, and the controller block. The application part and mode used are not part are usually not part of the BLE stack.

The Host contains the Generic Access Profile (GAP), the Generic Attribute Profile (GATT), the Attribute Protocol (ATT), the Logical Link Adaptation and Control Protocol (L2CAP) and the Security Manager (SMP), while the Link Controller, the Physical Layer (PHY) and the Direct Test Mode (DTM) are located in the controller. Between the controller and the host is the Host-Controller Interface (HCI).

The physical layer (PHY) refers to the radio hardware utilized for communication and for modulating/demodulating the data. The link layer is the layer that interfaces with the physical layer and provides the higher-level layers an abstraction and a way to interact with the radio.
DTM is only necessary for performing RF tests and is used during certification tests during manufacturing. The HCI layer is a standard protocol defined by Bluetooth specification that allows the host layer to communicate with the controller layer.

In the controller end, the L2CAP handles fragmentation and recombination, acting as a protocol-multiplexing layer. It takes the larger packets from the upper layers and splits them into chunks that fit into the maximum BLE payload size supported for transmission. On the receiver end, it takes multiple packets and combines them into one packet that can be handled by the upper layers. SMP is responsible for generating, storing and transmitting end-to-end cryptographic keys between devices.

The end-to-end protocol ATT acts as client-server, its role allows two devices to exchange commands, requests, confirmations, responses, notifications and indications that allow the reading and writing of values on an ATT server. GATT is an ATT interface through which the hierarchy of services, characteristics and attributes of a BLE device can be discovered. GAP provides the framework that defines how BLE devices interact with each other and is responsible for common features to all Bluetooth devices.

A. BLUETOOTH MESH
Mesh is a network topology used to establish a many-to-many wireless communication between devices. The mesh topology available in BLE enables the creation of long-range networks ideally suited to the Internet of Things (IoT). It is used to control, monitor and automate systems where tens, hundreds or thousands of devices need a secure way of communicating with others, enabling the creation of sensor networks.

One of the great advantages of Bluetooth mesh is the possibility of re-transmitting data to other devices that are not directly connected to the original device. In addition, it allows the coverage of very large areas, it enables the monitoring and control of a large number of devices with low power consumption, makes efficient use of radio resources and has compatibility with smartphones, tablets and products for personal computing devices.

Bluetooth mesh builds on top of BLE. It specifically utilizes the advertising state of BLE devices. Devices within a Bluetooth mesh network do not connect to each other like traditional BLE devices do. Rather, they use the advertising and scanning states to relay messages to each other. Figure 2 shows the layers of Bluetooth Mesh architecture.

1) Model layer: this layer addresses the implementation of models including behaviors, messages, states, and state bindings.
2) Foundation Model layer: this layer is concerned with network configuration and network management models.
3) Access layer: this layer defines how the application uses the upper transport layer.
4) Upper Transport layer: this layer is responsible for encryption, decryption, authentication and transport control messages.
5) Lower Transport layer: this layer handles two main tasks: the segmentation of packets from the layer above (upper transport layer) and reassembly of packets from the layer below (bearer layer).
6) Network layer: defines how messages are addressed and performs message forwarding, acceptance and disposal. Also defines the encryption and authentication mechanisms at the network level.
7) Bearer layer: this layer defines how the different mesh packets (PDUs) are handled.
8) Bluetooth Low Energy Core Specification: Bluetooth Core 5.0 specification.

The exchange of messages in this version of the BLE mesh network specification is done through a managed flooding mechanism. It has some differences in relation to conventional flooding. To limit retransmissions on the network, catching techniques are employed, where it is detected if the received message is duplicated. A time-to-live (TTL) mechanism is also employed, limiting unnecessary retransmissions.
security are performed independently and security keys can be exchanged during the lifetime of the mesh network.

The distribution of these keys and network addresses is done centrally through a mesh network provisioning mechanism, which is responsible for the formation of the mesh network and generation of unique cryptographic keys for the network and its applications. The provisioning process is used to add new devices to the network.

The Bluetooth Mesh Standard is defined as a publish/subscribe model where publishers can publish to a certain topic and subscribers can subscribe to one or more topics of interest. Publishing is the act of sending a message. Subscribing is a configuration utilized to allow select messages to be sent to specific addresses for processing.

To be able to connect these different publishers and subscribers, a mesh topology is created. The standard uses a BLE advertisement packet to communicate in a mesh network, which is only supported by devices that support both BLE and Bluetooth Mesh. Fortunately, the standard also defines a backwards compatibility feature to ensure that BLE devices which do not support Bluetooth Mesh can also be part of a Bluetooth Mesh network.

**B. FRUITYMESH**

In development by M-Way Solutions [7], Fruitymesh is an algorithm for the creation and maintenance of a mesh BLE network topology. The algorithm is based on neighbour-only routing, where no routing tables are stored but a connection is established between two nodes and kept open until the packets reach their final destination. The network is created through a clustering process and the path from source to sink is randomly formed.

The goal of the algorithm is not to provide routing but to build and maintain a topology that optimizes the BLE Mesh network. By providing a mechanism to update the topology in case of loss of nodes or in case of failure of nodes, the algorithm can be optimized for a number of use cases. Nodes can be inserted into the topology at different times and also after the route setup is finished.

The idea of Fruitymesh for topology assembly is that each node is always part of a cluster with a unique Cluster ID generated by the founding member. A single node without any connections represents a cluster with the size of one. Only nodes that are part of different clusters connect to each other and exchange their cluster information. This information is evaluated in the Cluster Score function. The node that is part of the bigger cluster can decide which connections it wants to make and will absorb nodes from smaller clusters into its own cluster. The resulting increased cluster size is reported back to the rest of the cluster. Size information is passed through the existing connections, which is an energy efficient way of distributing information. If nodes disconnect from a cluster either voluntarily or because of timeouts, the size of the cluster must be decreased in accordance with the number of nodes that were lost. This requires each node to save the quantity of connected nodes for each active connection.

Before a node can join a network, it must be assigned a Node ID and a Network ID. The Node IDs do not have to follow a scheme but must be unique for a particular network. The Network ID allows different networks to coexist in the same physical space. These values are configured in the enrollment process. After booting, a node generates a new Cluster ID. The initial Cluster Size of a node must be set to 1. A value called Connected Cluster Size must be saved for each possible connection (e.g., 1 outgoing, 3 incoming) and is initialized with 0. Figure 3 depicts a node that is directly connected with two other nodes and gives an overview of the stored values.

During the discovery phase, a node will constantly broadcast its Node ID, Cluster ID, and its Cluster Size in an advertising packet. This packet is called a JOINME packet (see Figure 4). It can include further attributes that other nodes used to calculate the Cluster Score. The advertising packet must be connectable if the incoming connection is still free, otherwise it is non-connectable.

After a predefined amount of seconds, a node calculates its best connection partner based on the collected JOINME packets. At first, it looks for nodes that have a freeIn connection (that has no master connected to it) connection and tries to connect as a master. Each packet is evaluated with the Cluster Score function, which returns a connection score. This score is zero if the node belongs to the same cluster or if the other cluster is bigger. After processing all packets, the algorithm tries to connect to the highest ranked node. In order to do this it needs to listen for another advertising packet that will answer with a connection request. If successful, the two nodes perform a handshake.

The ClusterScore function has fundamental importance for the Fruitymesh algorithm, as it will decide how the network topology will be mounted. All connections between clusters and the way in which the network will be mounted occurs due to the score returned by the function. This value can be
calculated in two different ways: There is with the score as master and the score as slave.

A node first tries to connect to another node as master. If this is not possible, the node will try to connect to another node as slave. For each JOINME packet received there will be a respective score. When attempting to connect as a master, the node will attempt to connect to the node that sent the packet that received the highest score.

Before the handshake is finished, the newly connected node cannot participate in the network and must not send any packets except those belonging to the handshake. Both nodes start the handshake by sending a CLUSTERWELCOME packet which includes the latest information about their Cluster Size and Cluster ID since the available information might already be outdated (see Figure 5).

After the network has been formed, the JOINME packets will all advertise the same Cluster ID. This will gradually cause all nodes to switch to a lower discovery mode when no changes are detected for a long time. They will then scan the network less frequently and send fewer advertising messages.

After a node is initialized, it enters the DISCOVERY_HIGH state in which it tries to connect to other clusters as soon as possible. After a timeout of a few seconds, it changes to the DECISION state where it decides whether it wants to connect to another cluster. If no other cluster is found, it sleeps for a few seconds in the BACK_OFF state to conserve energy. After several loops without finding another cluster, it will use the DISCOVERY_LOW state instead of DISCOVERY_HIGH. This allows the node to save more energy by scanning and advertising less. This is important when a node is switched on at a location without other nodes nearby or if the network has been fully discovered.

In the field of academic solutions, the existing proposals are divided between those involving flooding and those involving the use of some type of routing, static or dynamic. In each proposal, advertising channels, data channels or both were used to fulfill the purpose. Different platforms were also used and different metrics were evaluated. It is worth noting that in flooding method, the data travels only in the three advertising channels. In contrast, in some proposals with routing only information relating to routers is disseminated in advertising channels (via broadcast). Other data, as well as additional information about routes, travels in the other 37 data channels.

II. RELATED WORK
A. MESH NETWORK SOLUTION
Darroudi and Gomez [2] carried out an exhaustive research of the existing proposals until the

They provided a taxonomy for BLE mesh network solutions whereby they have divided the identified solutions into three main categories, namely: standardized solutions, academic solutions, and proprietary solutions.
In addition to these three parameters, Trickle maintains three run-time parameters: “I”, the current interval size, “t”, a time within the current interval, and “c”, a counter.

The Trickle algorithm has six rules:

1) When the algorithm starts execution, it sets I to a value in the range of [Imin, Imax] that is, greater than or equal to Imin and less than or equal to Imax. The algorithm then begins the first interval.

2) When an interval begins, Trickle resets c to 0 and sets t to a random point in the interval, taken from the range [I/2, I), that is, values greater than or equal to I/2 and less than I. The interval ends at I.

3) Whenever Trickle hears a transmission that is “consistent”, it increments the counter c.

4) At time t, Trickle transmits if and only if the counter c is less than the redundancy constant k.

5) When the interval I expires, Trickle doubles the interval length. If this new interval length would be longer than the time specified by Imax, Trickle sets the interval length I to be the time specified by Imax.

6) If Trickle hears a transmission that is “inconsistent” and I is greater than or equal to Imin, it resets the Trickle timer. To reset the timer, Trickle sets I to Imin and starts a new interval as in step 2. If I is equal to Imin when Trickle hears an “inconsistent” transmission, Trickle does nothing. Trickle can also reset its timer in response to external “events”.

Is important to emphasize that what constitutes consistency and inconsistency is defined on a case-by-case basis for each protocol that uses Trickle. That is, the specification of the protocol that uses Trickle must specify what is a consistent transmission and what is an inconsistent transmission. For the proposed work it has been defined a consistent message as one in which the node receives the same information as it did previously, and an inconsistent message as one in which the node receives information other than that previously received.

III. PROPOSAL OF A ENERGY BALANCED FLOODING ALGORITHM

Based on the Trickle algorithm, the Drypp algorithm was proposed. Drypp means Trickle in Norwegian. This name was given to the proposed protocol as a tribute to the device manufacturer, the Nordic manufacturer.

Trickle is based on the fact that all nodes are always watching and re-sending packets without any back offs or treatment for battery reduced consumption. Therefore, the BLE device runs out of battery power quickly. The proposed Drypp protocol seeks to change the behavior of the Trickle protocol so that the node’s battery consumption is reduced. This is achieved through intelligent control of message sending and re-sending intervals. This way you can achieve load balancing of nodes and avoid void zones.

A. DRYPP OPERATION

As mentioned in the previous section, in standard Trickle, if the design-time or run-time parameters have different values on different nodes, there will be consequences on the network. In a scenario with different values of Imax, nodes with lower values will transmit first, suppressing those with lower values, generating long-term inequities in energy expenditure and may cause the creation of so-called “void” zones, that is, regions in the topology where there is no node to relay data.

In a scenario where the nodes have different intervals, some nodes, when listening to inconsistent messages, will transmit before others. The faster nodes will see their intervals grow like the other nodes, but in this period slower nodes can be suppressed.

Unlike Trickle, network nodes no longer have to have the same values as design-time parameters, the Imin and Imax range limits, and the redundancy constant k. In the Drypp algorithm, the goal is to balance the consumption of nodes in the network. This makes nodes with less battery transmit less through longer back-off intervals. Therefore, the one with the biggest load will transmit more. The addition of two interval change constants, α and β, changes the value of the current Trickle interval I and the maximum interval Imax, respectively. The constants α and β aim to make the nodes with the largest battery have shorter I intervals, while the nodes with the smallest battery have longer intervals. This makes nodes that have less charge transmit less, saving its battery.

The values of I and Imax are derived from a value α and β for each battery range, this can be observed in table 1. It is worth remembering that Imax is not the maximum interval value that a node can have relative to a base value 2, i.e. if Imax = 3, the maximum Trickle interval duration in seconds is given by:

\[I_{\text{min}} \times 2^{\text{Imax}} = I_{\text{min}} \times 8\]

The value of interval I will be calculated from a constant α, which is calculated by

\[\alpha = 1 + \beta\]

with the next size of the Trickle interval calculated as

\[I = I \times \alpha,\]

while the value of Imax will be calculated as

\[\text{Imax} = \text{Imp} \times \beta,\]

where Imp is a constant with value 3. The redundancy constant k was set to 1 (one) because as the tests performed were in an environment without interference when a node hears that another node has already transmitted the packet, it immediately discards this packet from its buffer and the Imin is defined as the transmission time of a packet.

In the Trickle protocol the values of Imin, Imax and k are defined in an initialization phase, during interval I. The node will have a listening period between [0, I / 2] and a transmission period between [I / 2, I] at which time t will be the moment of transmission, it will be chosen randomly within the range (I / 2, I). Whenever the node receives a consistent message, in the case of Nordic nRF52, whenever
TABLE 1. Table of parameter values for each node battery range.

| Battery | $\beta$ | $\alpha$ | $\text{Imax}$ | $\text{Imax (s)}$ |
|---------|---------|---------|--------------|------------------|
| 100 - 75| 0.75    | 1.75    | 2.25         | 4.75 x $\text{Imin}$ |
| 74 - 50 | 1.00    | 2.00    | 3.00         | 8 x $\text{Imin}$ |
| 49 - 25 | 1.25    | 2.25    | 3.75         | 13.45 x $\text{Imin}$ |
| 24 - 0  | 1.50    | 2.50    | 4.50         | 22.63 x $\text{Imin}$ |

A message is received again by the node during the listening period, the redundancy counter $c$ will be incremented, and at time $t$, the message will be transmitted by the node only if $c \leq k$, where $k$ is the redundancy constant. If no message is received during the listening period, the interval $I$ value will be doubled. When the interval $I$ expires, it will be doubled until it reaches $\text{Imax}$ and so $c$ will be reset to the value 0. If a new message, different from the previous one is detected, an inconsistency will be found and the Trickle interval will be reset to $\text{Imin}$ and the whole procedure will be repeated.

As can be seen on figure 7 the Drypp protocol shows some differences in relation to Trickle, the detection of inconsistencies and the transmission mechanism remain the same, however, during the initialization phase, the constants $\alpha$ and $\beta$ will be defined from the node battery state, the constant of $k$ consistency is defined as 1 (one) and the Imp value defined as 3 (three). Imp is the base value based on which the $\text{Imax}$ value will vary. Every time the Trickle $I$ interval expires, $\text{Imax}$ will be calculated as $\text{Imax} = \text{Imp} \times \beta$.

Much of the Trickle structure is maintained at Drypp, but the biggest difference is seen when the $I$ interval expires. In Drypp, when the interval $I$ expires, in addition to resetting the redundancy counter $c$, the values of $\text{Imax}$ ($\alpha$ and $\beta$) can be updated if the battery range of the node changes. This mechanism is essential to save the battery life of a node in a balanced way. Unlike Trickle, as long as the interval $I$ does not expire, it will be multiplied by $\alpha$, instead of being multiplied by 2.

The Drypp algorithm follows these steps:

I. According to battery state and following the rules defined on table 1, $\alpha$ and $\beta$ are configured;

II. In the initialization phase, the values of $\text{Imin}$, $k$ and Imp are defined. $\text{Imin}$ is defined as the time that is spent to transmit 1 (one) packet, the redundancy constant $k$ is defined as 1 (one) and constant Imp with the value 3 (three);

III. Once initialized, the algorithm follows the standard procedures of the Trickle algorithm. The value of interval $I$ is defined as a random value within the range of $[\text{Imin}, \text{Imax}]$, the redundancy counter $c$ is reset and the transmission time $t$ is defined as a random value in the interval $[I/2, I)$;

IV. Just like the Trickle algorithm, whenever the node listens to an inconsistent transmission, that is, it listens to a transmission different from that received previously, $c$ will be incremented. The node will transmit the message on time $t$, if only if, $c$ is less than or equal to $k$;

V. Every time the interval $I$ expires, the node will increase the width of its interval to the value of $I \times \alpha$. If the new width of the interval is greater than the time specified by $\text{Imax}$, the value specified by $\text{Imax}$ will be defined as the size of the interval;

VI. When the interval $I$ reaches its maximum value, that is, the value specified by $\text{Imax}$, the battery state of the node will be checked again, and following what is shown on table 1 the values of $\alpha$, $\beta$ and $\text{Imax}$ will be updated. The timer will be reset, interval $I$ will be set to $\text{Imin}$ and counter $c$ will be reset, returning to step III.

IV. EXPERIMENTATION

During the experiment it was discovered that the Nordic Semiconductors Software Development Kit (SDK) version 14 for the nRF52 family utilized did not allow access Trickle code. The code was already inside the Softdevice. Due to this, the proposed protocol was not fully implemented as FruityMesh only supports the SDK version 14. Today the SDK is in its version 16 [6].

Access to the code is limited for some variables such as the transmission window size, transmission interval and waiting time before re sending, that is why the Drypp protocol was partially implemented. The Partial Drypp behavior can be seen in the flowchart specified in Figure 7.

The choice of the FruityMesh framework made by M-Way Solutions GmbH [7] was influenced by the fact that it has both...
protocols combined simultaneously in a hybrid protocol with balanced flooding (for broadcast information) and routing mechanisms of FruityMesh (for routing communication).

This hybrid architecture has a connection interval which specifies how often to synchronize with the peripherals. That interval can be divided into three states, each with different priority levels. These states are: connection events, listening interval and transmission interval. As can be seen in Figure 7, the first period, which is at the beginning of the connection interval, is reserved for packet exchange between the node and its masters and slaves (FruityMesh routing). The remaining time in the internal connection corresponds to the listening or transmission phase.

When a priority packet needs to be delivered, the listening interval is interrupted and switched to the transmission interval (Trickle). These packets use the Trickle algorithm and suffer minimal delay like upgrade commands, control packets, and JOINME packets. Since the priority is lower than that of the event connection range, the former cannot be interrupted.

The listening window is adjustable from a duty cycle. When set at 50% for that parameter, the connections event state of the device will immediately commence the listening phase and in the middle of that interval the node will turn off the radio and go into sleeping mode. This way, the device will be able to keep its power consumption low, increasing its service lifetime.

An example of the coexistence of both protocols in action is shown on Figure 8. Green blocks correspond to the communication mechanism between the node and its slaves and/or masters (topology configuration occurs in FruityMesh). The orange blocks represent the listening phase, while the light blue blocks represent the sleeping mode. Finally, the dark blue blocks are the Trickle transmission events, which occur from the interruption of the listening interval when it is required.

The modifications chosen within Fruitymesh is to change their values according to the state of the node battery. The variations are: the size of the window (that delimits the window in which the node can transmit within the total transmission window) and the advertising interval (because it is the controller of trickle message relay). The first value represents the result obtained from the Imin and Imax calculations and the second the advertising interval, which is the waiting time for sending messages from the network. The behavior was that expected from Drypp protocol and can be seen in Table 2.

Now it was possible to prepare a version of the Drypp protocol that fits the current code for concept tests. The algorithm of this version can be seen in Figure 7. The values exposed are multipliers of the parameter 0.625 ms which is used as a time base within FruityMesh both for Trickle control and for the protocol itself. The values utilized were obtained through the smallest windows in order to achieve a better scenario (the shortest possible delay).

This version simulates a low loss rate (because the values of the transmission window size and the waiting time for resending are small). This shows behavior very similar to the original protocol in the best-case scenario.

This algorithm was developed by changing the battery check function of the Status Reporting module. The edited file was the module code in the directory fruitymesh/src/modules/StatusReporterModule.cpp.

### Table 2. Data behavior in a factor of 0.626 ms.

| Battery [%] | Scan | Duty Cycle | Advertising |
|------------|------|------------|-------------|
| 100 - 75   | 20   | 5          | 150         |
| 75 - 50    | 20   | 4          | 600         |
| 50 - 25    | 20   | 3          | 1300        |
| 25 - 0     | 20   | 1          | 3600        |

### A. TOOLS

In this study, we used a set of hardware devices that can be divided in two groups: devices under study and support. The devices under study are the RedBear® Nano V2 development boards with the Nordic® nRF52832 microcontroller. The support devices are: DAPLink v1.5 (programming interface for RedBear® boards), a Nordic® nRF52 PCA 10040 board with a nRF52832 microcontroller used as a Bluetooth sniffer, a Lolin NodeMcu v3 board with an ESP12-E microcontroller and other auxiliary serial interface devices (FT232RL) for sending and reading FruityMesh states while running the transmitter (TX) and receiver (RX) nodes.

The RedBear Nano v2 was developed by RedBear. It uses the Nordic nRF52832 BLE SoC microcontroller that combines an ARM processor with a 2.4 GHz multiprotocol radio support and −96 dB RX sensitivity, 512 kB of flash memory and 64 kB of RAM. It works with Bluetooth low energy versions 4.x and 5.0. RedBear is programmed through the DAPLink v1.5 interface [11], which allows the insertion of already compiled code in HEX format. The programming of this device is done in several IDEs, mainly Arduino [8], Keil µVision [9] and Segger [10] using the open source SDK provided by Nordic for development in their devices.

The SDK can be used directly through the GNU-ARM-GCC compiler [12], GNU-MAKE. In the case of the study, the compiler required by Fruitymesh is the GNU ARM Embedded Toolchain in version 4.9 [7]. The proper settings to run Fruitymesh are edited in the makefile.local files.
(specifying the Toolchain location, the platform used - nRF52 - and the debug mode, release). The choice of platform is defined by the chip used. In this case, the nRF52832 uses SDK 14 with the Bluetooth 4.2 BLE.

In addition, the Makefile.nRF52 file was mapping the start addresses of the RAM memory and its size was corrected according to the SDK and softdevice used in Fruitymesh. After the make command generates the compiled file, it is necessary to add it to the pre-compiled Bluetooth protocol stack called softdevice, provided by Nordic, through the MergeHex tool. Then that HEX file is placed on the RedBear device through DAPLink that allows an interface of USB Storage type. It allows one to drag and drop the file into a folder after connecting the DAPLink to the computer.

FruityMesh requires the softdevice version 5.1.0. But in this version, Trickle is configured inside the stack without the possibility of changing its code. However, in the current version of Nordic’s SDK, the Trickle code is described in a file of its own.

The PCA10040 is used as a Bluetooth BLE Sniffer with Wireshark. The PCA 10040 has a special program which allows its to works like a Bluetooth Sniffer through a USB interface into Wireshark. It is necessary to add a plugin to be able to understand the Bluetooth Mesh packages and understand the interface of the PCA 10040 with the sniffer program installed as a source of capture [6].

Three ESP12-E NodeMCU v3 were used to record the battery of each node in real time. This device has a 32-bit Esp8266 microcontroller and an integrated WiFi interface. Its programming can be done by the Arduino IDE with the addition of specific libraries to it. The microcontroller analog input that supports input voltage of 1.1 V. This required the use of a voltage divider that divides the 3.3V from the maximum battery to the allowed 1.1V. This divider was made with the association of 3 resistors of 1 KΩ in series with a pair in parallel which divides by three the voltage at the junction between the resistor alone to the pair in parallel. This value is converted by a 10-bit A/D converter and gives an accuracy of approximately 1 mV. The microcontroller has been programmed to behave as a simple http server. It then takes the data from the AD converter and passes it to an html page within the local network to every device connected. This page gives the node ID, time and battery voltage and is saved through a python script every 10 seconds. The device saves the data displayed in a csv file (Comma-separated values) [14]. Thus it makes it possible to observe the behavior of the batteries in each node without sending packets through BLE.

The Serial USB interfaces is necessary to see the status of the network and if it was correctly propagated. Its function is to open a serial terminal to the end devices (transmitter and receiver) in order to allow seeing the network and send commands from one end to another. The default FruityMesh setting is 100000 bauds. Its is used on the ports configured in the board_4.cpp file and for the run of commands and to receive log updates from devices. It also determines which log shows the size of the network, how many are connected, battery value and other data.

It was necessary to use two computers to check the network by connecting each RedBear from the ends through a Serial USB interface to one of the computers. A notebook and a Raspberry 3 remotely accessed via ssh in the local network were used to control the two external nodes of the network through one computer.

One of the points observed was the consumption of the battery. But using normal coin batteries would make it so long that it could take days to run a single test. Thus, the use of supercapacitors that enable a fast charge (10 seconds connected to a source of high current charge) and a similar charge every time was proposed: TCAM model 8QQ0302 10F 2.7V capacitors [15].

B. PROCEDURES OF EXPERIMENTS
Initially, the study was conducted on the Fruitymesh and Trickle protocols. Then the Drypp proposal was created, the modified Trickle for development of the proposed solution using the tools and devices presented in the last section.

The configuration of RedBear nodes (pins and settings referring to the used board itself) was made inside the config/boards folder where it can find the default settings supported by Fruitymesh and the space to add more boards. However, since Redbear uses the same BLE integrated circuit as the PC10040 card (the sniffer used), just some pins and others simple settings had to be adjusted in the board_4.cpp file.

The platform settings, build type and GCC ARM path are defined in makefile.local. Card transmit power settings were set to −20 dBm by changing the setting in fruitymesh/config/sdk_config.h by setting ANTFS_CONFIG_TRANSMIT_POWER to 0 (zero) which sets the transmit power to −20dBm.

The ping module has been added for testing purposes. This module sends pings at a regular rate through a command sent through the serial interface. Pings can be sent either routed by Fruitymesh (when assigned to a node) or broadcast (sent to node 0). When sent in broadcast form, the behavior was modified so that a large number of pings would be sent followed with a fixed interval of 5 seconds (this interval for reading sensors of temperature). The module is the same in the two protocol tests. Being an important traffic generator, however, the continued sending can overload the device buffers and no buffer cleaning was implemented. That module was created based on the example proposed in the Fruitymesh [7].

The type of the node that will be the sink, i.e. the target node is setting up fruitymesh/src/config.cpp file and changing the deviceType statement as a deviceTypes::DEVICE_TYPE_STATIC to a deviceTypes:: DEVICE_TYPE_SINK. This finalizes the basic configuration for both Fruitymesh Trickle and Fruitymesh Drypp.

The details of what was edited from Trickle to generate Drypp are detailed in the algorithm shown in Figure 7. To configure Drypp it was necessary to
discover a point where the code could be changed. The file that contained the 3 parameters of Figure 7 was the fruitymesh/src/modules/StatusReporterModule.cpp file. The modifications were made within the convertADCtoVoltage(i16*buffer, u16 size) function with the addition of 4 variables to the scope of the module as a whole. One to save the battery current state and another 3 for the compliant battery temperatures. But the measure needed a normalization in order to actually observe the state change in a similar way on all devices.

The code added to the battery measurement function has a particularity in that in the measurement it requires the normalization of the reading in order to express the decay of the battery not as a non-variable value. This is due to the measurement of the Vdd pin being made from its reference which causes variation in the reading for different battery values [13]. After some tests the battery decay was checked in 1-digit steps for the AD converter settings determined in the board4.cpp. The behavior was verified by activating the StatusReport module packet that reads every 60 seconds and the capacitor charged the Redbear for about one hour and 15 minutes.

The capture of the values of one of the tests of the return of the battery value through the device demonstrates the behavior was that the battery value of the RedBear node. At first, it was observed a very high value, the first measurement of the device returned 185 (battery value is expressed in a byte without signal). In the second measurement, the value obtained was 143. The next collections maintained the value and followed the decay of the battery, changing to 142 when the battery reaches around 75% of charge, 141 when reaching 50%, 140 to 25%, sometimes even showing the value 139 or the battery enters the voltage below the minimum that the device needs.

So the idea of normalizing the battery designed in order to create a variable of the battery state based on intervals. The first measurement with high value - Bat1 stores this initial value - represents the state 0 (zero). After receiving a stable value after the state becomes 1, it stores the new battery value in Bat2 and increases the state. Thus, it waits for the next battery value that is saved in Bat3 and compares if it is smaller than Bat2 to verify if it changed state or not, increasing by 1 the state if possible. The proposed solution for the normalized capture can be seen in the flowchart of Figure 9. The variable state is important in order to verify in which battery level the node is. In 3 it is between 100% and 75%; 4, between 75% and 50%; 5, between 50%, 25%; and 6, between 25% and 0%. Note that 0% of battery is when the battery voltage reaches 1.7 V, minimum functional voltage for RedBear works on the vdd port [13].

The values of the other tests were quite different, but the decaying behavior was similar. The normalization can be summarized in the flowchart of Figure 9.

With the modified code, three versions of the code were created to be used in the experiments. The trickle version (without the addition of the Drypp code), the Trickle sink (without the addition of the Drypp and with the node seen as the destination) and the Drypp version (with the addition of the modification and only used in the middle nodes of the network).

The tests were performed in the following order and topologies: Tandem Trickle, Tandem Drypp, Parallel Trickle and Parallel Drypp. The execution procedure is expressed by the following steps:

1) Compilation of the Fruitymesh code and its merger with a compatible sofdevice;
2) Insertion of the program to be tested in the central devices through DAPLink;
3) Assembly of battery monitoring systems;
4) Positioning of network nodes according to the proposed topology;
5) Charging the capacitors to start the network;
6) Initiating the capture of battery data and monitoring of serial interfaces;
7) Starting the capture of BLE packages through Sniffer;
8) Recharging of batteries from central nodes up to 3V;
9) Waiting for the network to lose connectivity;
10) Finalizing the collection programs and analyzing the data.
Step 3 was the connection between the capacitor used as a battery by Redbear and the analog input of the Esp12-E board so that it could have access to the battery value.

The data capture of the batteries, step 6, was done through a python script which ran for the IP given to the device within the local network. Three instances were used to capture the data referring to the three central nodes. The FT232RL devices were used for monitoring the serial interface of TX and RX devices. The serial interface of the Arduino program configured at 1000000 bauds on the computer to read the RX and, via SSH, the computer connected to Raspberry and executed the command cu -l /dev/ttyUSB0 -s 1000000 that simulated a serial interface through the terminal to access the TX.

After the capture of battery data is started, the Sniffer BLE capture was also started through Wireshark. However, for wireshark to see BLE packets it was necessary to add the BTMesh plugin to inspect the mesh packets on Wireshark.

Before performing step 8 it was verified whether or not the mesh network was created. If the nodes were properly forming a network, the logs in the serial terminals would show “cluster size = 5”. Then all capacitors of the central nodes were charged one by one, leaving all capacitors above 3V for the lifetime of the network.

During step 9 Ping commands (pm 0) were made through the TX terminal to send a broadcast ping to the network every 5 seconds. In addition network control data was also sent via broadcast. Eventually, we waited until the central nodes ran out of battery and RX and TX nodes could not communicate anymore.

Step 10 was the collection time. The pcaps (Wireshark extension), CSVs and Logs files were saved to the comparison of the Trickle and Drypp behaviors in the two topologies under study.

C. RESULTS

The primary objective of this paper is to evaluate the difference in energy expenditure between Trickle and Drypp and the consequences generated by the operating mechanism of the proposed protocol. The lifetime of the first node of the network to run out of battery, the lifetime of the network, the throughput and the number of lost packets are the metrics of this study.

The comparison between the two protocols was made with the use of two different scenarios: one scenario where data was first collected from the nodes and network using Trickle protocol; the other where data was collected from the nodes and network using the Drypp protocol.

In figures 10, 11, 13 and 14, Node 1 is the black line, node 2 is the green one and node 3 is the orange one. The circles represent the batteries’ state change. Green is 100%, light green is 75%, orange is 50% and red is 25%.

1) TANDEM SCENARIO

This scenario was used to define a maximum range topology in which all nodes must be active for TX to communicate with RX. The topology was performed inside a house with a distance of 1.5 meters among nodes.

The results of the packet flow of the middle nodes (n1, n2 and n3) are shown in Figures 10 and 11. In these graphs it is possible to observe a similar behavior of the throughput of the nodes in the two protocols. But certain stabilization is seen on the Drypp graph (Figure 11) that keeps the throughput values more stable starting from 1000 seconds onwards. The stabilization does not occur in Trickle (Figure 10).

In summary, Table 4 outlines all the important data obtained in the experiment. The average throughput is below the gain in network life. This occurred because the network sent fewer packets which increased the chance of a successful sending. This consequently reduced losses and retransmissions.
In the parallel scenario it is important to see whether the behavior of the load balancing between the nodes is occurring or not. Thus, it is possible to verify if Drypp forces the balancing of the middle nodes.

Only comparing the Throughput of Drypp and Trickle (Figures 13 and 14 respectively) it is possible to see the packet flow converging to the same number of packets per second in Drypp protocol.

Regarding Figures 16 (battery by time in Trickle) and 15 (battery by time in Drypp) the difference between the fall time of each node is presented in a divergent way (increasing) in Figure 16 but convergent way (reducing the difference) in Figure 15. This is also observed in Table 5. In Trickle, the difference of death from the first node to the last is 9 minutes and 9 seconds. But in Drypp, this difference falls...
to 2 minutes and 35 seconds (see Table 6). This shows a reduction of 28% from the time the first node turns off until the last node turns off.

V. CONCLUSION AND FUTURE WORK
As one of the largest low-power wireless technologies BLE technology faces significant challenges. One is its limited coverage in its standard star topology. With the advent of Bluetooth mesh several solutions have been proposed to increase the range of BLE networks and add support for more industrial applications using BLE technology.

One of the simplest methods of communication used is flooding as it does not require routing protocols, avoids delays and complexity. However, it can be potentially inefficient in one-to-one communication in large networks and, most importantly, it can be inefficient in terms of energy consumption.

With Drypp a BLE mesh network protocol solution was proposed. This protocol would save the battery of the nodes present in the network, balancing their loads of flooding changing messages and forming an intelligent topology using FruityMesh. The Trickle and the flooding protocol can work together.

This study used a simple test with a ping application showed that the Drypp protocol is more efficient than the Trickle protocol for the delivery of flooding messages. One of the biggest drawbacks in using the native form of Trickle could be the high power consumption, which would be harmful to a low power network.

From table 3, it can be seen that in the tandem topology scenario there was an increase in network lifetime. Therefore the first node to die had a longer lifetime, resulting in an increase in network lifetime from 51 minutes and 42 seconds to 57 minutes and 24 seconds, an 11% increase in network lifetime. On the other hand, the average throughput of the network decreased by only 7.5%.

It was observed that the lifetime of the network had very similar values when comparing the two protocols in parallel topology. This was due to the variation of the capacitor load and the unbalanced load in Trickle. However, when the node load balancing occurred in the Drypp scenario, the battery was balanced and the node deaths were very close to each other, differently from the case of the Trickle protocol.

One of the biggest challenges was the lack of available information and documentation in relation to FruityMesh but as it is an open source project there is still a vast path that can be covered regarding its functionality. Overall, it is a promising model that enables a balanced load consumption by using not only Drypp protocol, but also by sending data in a routed form that avoids re-transmissions. One of the limitations found was the access to the standard Trickle parameters that were inside the softdevice used. But, with the new SDK 16, it will be necessary to make adaptations in FruityMesh and make it compatible with the new SDK.

The most evident proposal for future work is the implementation of the complete Drypp protocol and its improvement, because the project was limited by the SDK version 14. But with the SDK version 16 it was observed that all values and parameters of Trickle can be easily found and edited in the available code.

REFERENCES
[1] M. Baert, J. Rossey, A. Shahid, and J. Hoebeke, “The Bluetooth mesh standard: An overview and experimental evaluation,” Sensors, vol. 18, no. 8, p. 2409, 2018.
[2] S. Darroudi and C. Gomez, “Bluetooth low energy mesh networks: A survey,” Sensors, vol. 17, no. 7, p. 1467, 2017.
[3] M. Afaneh, “Intro to Bluetooth low energy,” Novel Bits LLC, Tech. Rep., 2018.
[4] Y. murillo, B. Reyenders, A. Chiumento, S. Malik, P. Crombez, and S. Pollin, Bluetooth Now or Low Energy: Should BLE Mesh Become a Flooding or Connection Oriented Network. Brussels Belgian: Leuven & Televic Healthcare, 2017.
[5] The Trickle Algorithm: Issues and Solutions (Version 2.0). Reino Unido: Centre for Electronic Warfare, Cranfield University, Bedford, U.K., 2015.
[6] Nordic Semiconductor. NRF52832 Product Specification v1.4. Accessed: Jul. 2019. [Online]. Available: https://infocenter.nordicsemi.com/pdf/nRF52832_PS_v1.4.pdf
[7] (2019). M-Way Solutions GmbH. FruityMesh—The First Completely Connection-Based Open Source Mesh on Top of Bluetooth Low Energy 4.1. Germany. Accessed: May 2019. [Online]. Available: https://github.com/mwaylabs/fruitymesh/wiki
[8] RedBear. (2017). Nrf5x. Github. Accessed: Mar. 2019. [Online]. Available: https://github.com/redbear/nrF5x
[9] Armkeil. (2018). Keil µ Vision IDE. Accessed: Apr. 2019. [Online]. Available: http://www2.keil.com/mdk5/uvision
[10] A Complete All-In-One Solution, SEGGER Microcontroller GmbH, Hilden, Germany, 2018.
[11] Nano 2 & Blend 2. RedBear. (2016). Red Bear Company Limited. Bluetooth 5 Ready: BLE Module. Accessed: Mar. 2019. [Online]. Available: https://www.kickstarter.com/projects/redbear/Bluetooth-5-ready-bluetooth-module
[12] GNU MCU Eclipse. (2018). GNU MCU Eclipse ARM Embedded GCC v7.2.1-1.1 20180401 Released. [Online]. Available: https://nongnu.org/gnuarmemu/gnuarmemu.html
[13] Nordic Semiconductor. NRF52832 Product Specification v1.4. Accessed: Mar. 2019. [Online]. Available: https://infocenter.nordicsemi.com/pdf/nRF52832_PS_v1.4.pdf
[14] Network Working Group. RFC 4180 Common Format and MIME Type for Comma-Separated Values (CSV) Files. Accessed: Apr. 2019. [Online]. Available: https://tools.ietf.org/html/rfc4180
[15] Aliexpress. Super Capacitor 10F. Accessed: Aug. 2019. [Online]. Available: https://pt.aliexpress.com/item/32872095626.html
[16] D. Sundaram, “Green computing turns green IT.” Int. J. Advance Res. Comput. Sci. Manage., vol. 4, pp. 204–208, Apr. 2016.

ADRIANO SOUZA BRANDÃO received the bachelor’s degree in accounting from the University Central of the Federal District (UDF), in 2016, and the degree in communication network engineering from the University of Brasilia (UnB), in 2019.

He currently participates in the research group on self-adaptive systems with the Department of Computer Science, UnB, with Professor Drª Genaina. He works in the shared hosting management part of Site Hospedagemgrats.net. His research interests are in the area of Internet of Things (IoT), Bluetooth and wireless communication, self-adaptive systems, and Linux.
MATHEUS CORDEIRO LIMA received the bachelor’s degree in communication network engineering from the University of Brasilia (UnB), in 2019. He was a Volunteer Professor of electronics in public schools for the Electron Project of UnB. He also participated in the project of systemic simulation of satellite networks with the company Autotrac and UnB. His research interests are in the area of telecommunications, wireless communication, and is also interested in teaching.

CLAUDIA JACY BARENCO ABBAS received the degree in computer science from the Universidade Católica de Brasília (UCB), Brazil, in 1990, the M.Sc. degree in computer science from the Universidade Federal de Santa Catarina (UFSC), Brazil, in 1995, and the Ph.D. degree in telecommunication engineering from the Universidad Politécnica de Madrid (UPM), Spain, in 2000. She was a Teacher with the Universidade Católica de Brasília (UCB), from 1992 to 2000. Additionally, she was a Professor with Universidade Simón Bolivar (USB), Venezuela. From 2009 to 2011, she worked on research projects with the Arab Countries League, Damascus, Syria. From 2011 to 2013, she worked on a software development project with the Community of Latin American and Caribbean Countries (CELAC), Caracas, Venezuela. She is currently a Professor with the Department of Electrical Engineering, Universidade de Brasilia (UnB), Brazil. Her main research interests include wireless sensor networks, the Internet of Things (IoT), routing protocols, and performance of networks.

LUIS JAVIER GARCÍA VILLALBA received the degree in telecommunication engineering from the Universidad de Málaga, Spain, in 1993, and the Ph.D. degree in computer science from the Universidad Politécnica de Madrid, Spain, in 1999. He was a Visiting Scholar with the Computer Security and Industrial Cryptography (COSIC), Department of Electrical Engineering, Faculty of Engineering, Katholieke Universiteit Leuven, Belgium, in 2000, and also a Visiting Scientist with the IBM Research Division (IBM Almaden Research Center), San Jose, CA, USA, in 2001 and 2002. He is currently an Associate Professor with the Department of Software Engineering and Artificial Intelligence, Universidad Complutense de Madrid (UCM), and also the Head of the Complutense Research Group GASS (Group of Analysis, Security and Systems) which is located in the Faculty of Computer Science and Engineering, UCM Campus. His professional experience includes the management of both national and international research projects and both public (Spanish Ministry of R&D, Spanish Ministry of Defence, Horizon 2020—European Commission, . . .) and private financing (Hitachi, IBM, Nokia, Safelayer Secure Communications, TB Solutions Security, . . .). He has authored or coauthored numerous international publications and is an Editor or a Guest Editor of numerous journals such as Entropy (MDPI), Future Generation Computer Systems (FGCS), Future Internet (MDPI), the IEEE Latin America Transactions, IET Communications (IET-COM), IET Networks (IET-NET), IET Wireless Sensor Systems (IET-WSS), the International Journal of Ad Hoc and Ubiquitous Computing (IJAHUC), the International Journal of Multimedia and Ubiquitous Engineering (IJMUE), the Journal of Supercomputing, and Sensors (MDPI).