A Multiprotocol Low-Cost Automated Testbed for BLE Mesh

Yuri Murillo, Brecht Reynders, Alessandro Chiumento and Sofie Pollin
{yuri.murillo, brecht.reynders, alessandro.chiumento, sofie.pollin}@esat.kuleuven.be
KU Leuven, Department of Electrical Engineering (ESAT), Kasteelpark Arenberg 10, 3001 Heverlee, Belgium

***Abstract*—Bluetooth Low Energy (BLE) is envisioned as one of the most prominent technologies for the Internet of Things (IoT). The recent release of BLE mesh has attracted interest from both the industrial and academic communities, with new proposals for optimizing its performance being published on a regular basis. However, the majority of these publications rely on analysis and simulations, potentially neglecting the effect of real life environments and hardware limitations. Moreover, no framework is available to experimentally compare the performance of different mesh protocols on top of BLE. This paper presents the design and implementation of a low-cost, modular and fully automated testbed capable of running multiple mesh protocols. It is also remotely accessible for any third party interested in performing network experiments with it. Apart from discussing the testbed architecture, we provide results obtained from a measurement driven comparison of two protocols that fit within BLE operation: flooding and connection based. These results show that there is a trade off between the two approaches and that the optimal mesh protocol to use is application dependent, which should be taken into consideration when heading towards the next release of BLE mesh.**

I. Introduction

The Internet of Things (IoT) presents a paradigm shift in which wireless networks are expected to become densely populated. Industry leaders estimate that by 2021 there will be 11.6 billion wirelessly connected devices in the world, a figure 1.5 times greater than the projected global population [1]. Such dense networks introduce several challenges in terms of interference, power efficiency, mobility and hierarchical organization [2].

Wireless Mesh Networks (WMN) are emerging as an adequate solution: large areas can be covered by adding more low power devices that forward packets to the destination, creating ad-hoc networks that guarantee reliability by using alternate paths and channels. This approach becomes particularly interesting for well known, widely spread and cheap short-range technologies like Bluetooth Low Energy (BLE). In fact, enabling mesh operation in BLE has recently attracted much interest, with proprietary protocols released by industry [3] and approaches proposed by academic research [4], [5]. This process lead to the release of the first official BLE mesh specification in July 2017 [6].

Being such a novel technology, the number of publications and use cases for BLE mesh at the time of writing this work is very limited. Moreover, the majority of them focus on theoretical analysis and simulations [5], with few papers presenting experimental work [4]. As a result, there is no clear knowledge on the real performance of the protocol or its main aspects that should be optimized. In order to explore these uncertainties we implemented a multiprotocol BLE mesh automated testbed. Its multiprotocol nature means that several mesh protocols can run simultaneously, not being limited to just standard BLE mesh. This way, both connected and flooding based mesh protocols can be characterized. The automated nature allows to perform large batches of unsupervised experiments. The framework proposed can inspire researchers to design and implement their own wireless networking testbed based on the choices hereby presented. Although previous work has been done on building custom BLE mesh protocols and testbeds [4], to the best of the authors' knowledge there is no other current testbed implementation capable of running multiple BLE mesh protocols with the aim of experimentally comparing them.

The proposed architecture presents several strong aspects that help overcome the limitations of analytical and simulation methods:

- Using off-the-shelf nodes guarantees that the performance results obtained are representative of any real life application using BLE mesh networks. Moreover, it provides insight into additional issues, such as interference, clear versus obstructed environments, varying network load, etc.
- Support for several protocols allows comparison of different mesh flavors against regular BLE mesh. Real-time network behavior can be tested and strong and weak aspects of every protocol depending on the application targeted can be extracted. This serves as a powerful tool to further taylor the BLE mesh specification in the future.

Additionally, the design and implementation choices presented are chosen to solve the main impediments for network testing and testbed evaluation:

- A modular design along with a set of single-board computers, off-the-shelf nodes and open-source software makes the testbed scalable and reduces its overall cost.
- All network parameters, topologies, parsing and data storing functions can be pre-configured. These configuration files can be reused and stored as templates for generating new tests. This workflow simplifies the task of setting up an experiment, often a tedious and time-consuming process.
- Once the network tests are configured, any number of them can be programmed and performed automatically. Therefore, a large batch of experiments can be conducted

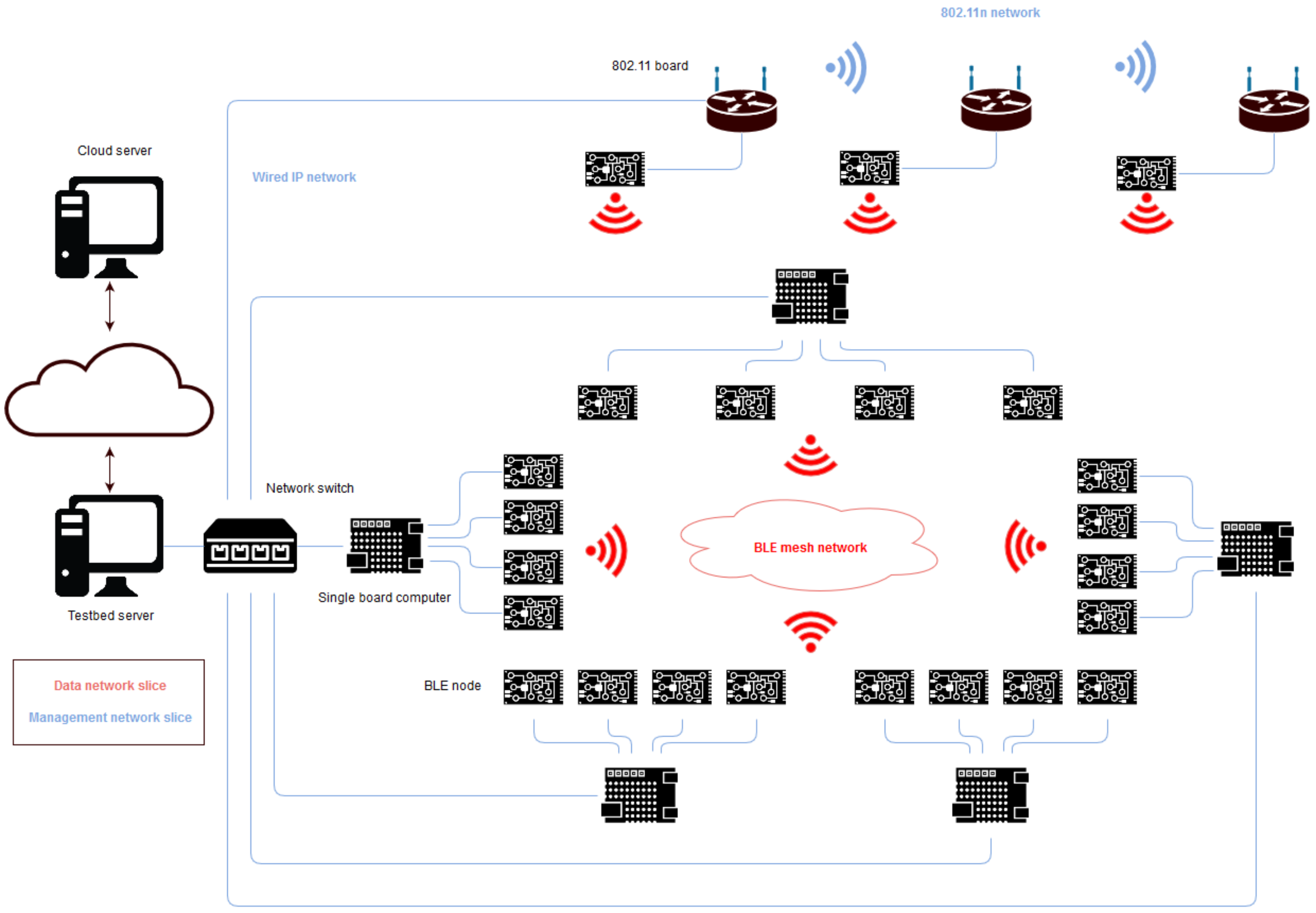


Fig. 1: Overview of the architecture of the testbed. The BLE mesh network becomes the data network slice, while the management network slice is subdivided into two parts: a wired IP network and a 802.11n network. The singe board computers and 802.11 boards are the main elements of the management network slice, which connect to the testbed server through a network switch. The PC acts as database, status visualization platform and gateway. A cloud server is used for remote login and backup purposes.

without intervention of the researcher, resulting in a more time efficient and less error-prone process.

- Full integration with state of the art platforms for testbed remote access [7]. Operators of the testbed have the possibility of remotely login to it and manage the experiments. This feature is completely free and available to any third party interested.

The rest of the paper is structured as follows. Section II describes the design and implementation of the testbed in fine detail. Section III introduces a case study of its use for a comparison of BLE mesh protocols, and finally Section IV concludes this work.

II. BLE Testbed: Design, Implementation, Features and Configuration

This section presents a description of the architecture and hardware used, an introduction of the several protocols supported and an overview of the software developed.

A. Architecture and Layout

Figure 1 depicts the architecture of the testbed. It consists of a set of BLE nodes, single-board computers, 802.11 boards, a network switch and a local and cloud PC servers. The main reason behind the proposed design choice is ensuring modularity of the testbed.

Two different network slices are defined: management and data. The management slice is subdivided into two parts: a wired IP network, used to control and establish a serial communication with the BLE nodes; and a 802.11n network, used to seamlessly access mobile nodes located far away from the testbed and disconnected from the wired backbone. The data network is the short-range mesh network established by the BLE nodes.

Although the current protocol developed for BLE supports individual node addressing and over-the-air parameter configuration, this task is only performed through the wired or 802.11n management slice. Therefore, control packets are sent over this network slice while data packets are sent over the data network slice. The former are used to configure the nodes and report their status, while the latter are used to send actual data from the sources to the sinks. This separation ensures that both types of traffic do not conflict with each other.

The testbed is spread over two adjacent rooms in a regular office environment. The main room is 6x9m and allocates 12 BLE nodes, one 802.11 board (with an additional BLE node) and the PC server. The second room is 4x8m and allocates 8 BLE nodes and one 802.11 board, plus its additional node. The last 802.11 board and node are located in a separate office, 30 meters away from the testbed. The Odroids and nodes are hanging from rails 2.5m above the floor creating a grid where the minimum separation between two nodes is 2m. The nodes are placed facing down, although their antennas are omnidirectional. There are no WiFi access points in neighboring rooms and the only 2.4GHz traffic is the one generated by the testbed. For this reason no interference controlling mechanisms are specifically designed, relying on 802.11 CSMA/CA and BLE AFH to ensure the coexistence between the data network slice and the wireless management slice.

B. Hardware

1) BLE nodes: The BLE nodes chosen are the Nordic Semiconductor nRF52 development boards [8]. They have four LEDs and four buttons which are user-programmable; several I/O interfaces available via connectors; support for Segger J-Link OB; an integrated PCB antenna; a coin-cell battery and a micro-USB connector for programming, powering and UART serial port communication. The chip has a 32 bit ARM Cortex M4F processor with 512 kB of flash memory and 64 kB of RAM.

The BLE nodes are the only elements in the data mesh network. They are flashed with a BLE stack and the mesh protocol is developed as an application on top of it. Every four nRF52s are connected to a single-board computer through their USB connection for powering and terminal communication purposes.

2) Single-board computers: Five Hardkernel Odroid-C2 single-board computers [9] have been selected. Each one has four 64 bit ARM Cortex A53 processors; a Mali 450 GPU; 2 Gigabytes of DDR3 SDRAM; 40 pin GPIOs; an HDMI port; four USB 2.0 host ports plus an additional OTG miniUSB for power and data; a Gigabit Ethernet port and an eMMC5.0 HS400 / UHS-1 SDR50 MicroSD flash storage slot. We opted for the eMMC card due to its faster speed and overall higher performance, which is flashed with Ubuntu MATE 16.04 LTS.

The Odroid is the main element of the management network: it runs the software needed for executing experiments and establishes a UART session with each of its four connected nRF52 nodes. When a node receives a mesh packet it sends it over the UART to its corresponding Odroid, which parses it into JSON format and logs it on the server database through the Ethernet management backbone.

3) 802.11 boards: Three MikroTik RB912UAG-2HPnD Routerboards [10] are used as 802.11 boards, which are small integrated routers capable of operating in 2.4 and 5 GHz. They have an Atheros AR9342 chipset at 600MHz; 64 MB of RAM; 128 MB of NAND flash memory; support for 802.11 b/g/n; 1 Ethernet port for traffic and PoE and finally a USB port. They are flashed with OpenWrt/LEDE 17.01.4, a Linux distribution targeted for embedded devices.

Component	**Unit cost (€)**
23x nRF52 BLE Development boards	37
23x microUSB cables	2
10x USB extension cables	3
5x Odroid-C2 single-board computers	45
5x Power supply plugs	4
5x Odroid cases	3
5x 32 GB eMMC modules	35
3x RouterBoard 802.11 boards	85
3x PoE injectors	14
6x Omni antenna 2.4 and 5 GHz	6
1x Netgear Network Switch	70
4x 20m Ethernet cable	6
TOTAL:	€1789

TABLE I: Testbed components and price. The cost of the server PC is neglected, as any existing office computer is powerful enough to run the software for the testbed.

The three 802.11 boards create a long range mesh WiFi network using B.A.T.M.A.N. Advanced, a linux kernel module running the B.A.T.M.A.N. layer 2 routing protocol [11]. It emulates a virtual network switch which encapsulates and forwards all traffic to the destination.

A BLE node is connected to each 802.11 board through USB. However, in this case a Serial to Network Proxy (ser2net) [12] session is configured in the Routerboard, which allows to establish a remote TCP connection and access the UART serial communication of the BLE node. Then, in the server the linux *socat* utility is used to establish a bidirectional byte stream to this TCP session. As a result, the BLE nodes attached to the 802.11 boards are seamlessly accessible from the server and appear as if they were directly connected to it. This provides a management link to these mobile nodes that can be far away from the testbed, and since the three Routerboards create a mesh network only one of them needs to be directly connected to the server.

4) Network switch & Server: A Netgear GS108T 8-port switch connects the Odroids to the server, which is a regular PC running Linux Mint 18.3 Sylvia. It has three main roles:

1) Control server to configure, run and visualize network experiments in real-time.
2) Database to log packets sent and received over the mesh network along with metadata extracted from them.
3) Gateway to allow internet connectivity on the Odroids and to push data to the cloud.

The server is the central point and the user interface of the testbed. Therefore, it is critical that redundancy and backup mechanisms are deployed in order to allow remote access and to ensure minimal data loss in case of failure. In this case, all mesh traffic is sent both to the testbed server and to a cloud server for redundancy. Additionally, a backup of the database is pushed to it on a daily basis.

Having this secondary server allows to specify two types of users: operators and administrators. Operators may access the cloud server remotely to get real-time information and

Settings

Field	Comment
TEST_ID	Test identifier
REALIZATIONS	# repetitions/test
PACKETS	# packets/repetition
SOURCE_RATE	# packets/s
SIZE	Size of the network
POWER	TX power
CONINTERVAL	
OUTCONNECTIONS	
MODE	[BLE/Trickle/FM]
DATE_START	
DATE_END	
DONE	Flag for test finished
OBSERVATIONS	User comments

Roles

Field	Comment
TEST_ID	Test identifier
NODE	Odroid
DEVICE	BLE node
ADDRESS	FM address
ROLE	[Source/Sink/Mesh]

Nodes

Field	Comment
TEST_ID	Test identifier
NODE	Odroid
DEVICE	BLE node
ADDRESS	FM address
READY	BLE node ready
BUILT	Flag network built
FINISHED	Flag repetition done

Control

Field	Comment
TEST_ID	Test identifier
NODE	Odroid
DEVICE	BLE node
ADDRESS	FM address
REALIZATION	Run identifier
NID	TX address
PID	Packet identifier
INCONN	(Address, RSSI, queue)
OUTCONN1	(Address, RSSI, queue)
OUTCONN2	(Address, RSSI, queue)
OUTCONN3	(Address, RSSI, queue)
BATTERY	Battery level
CONINTERVAL	
TX_POWER	
DEVICE_TYPE	[Sink/Static/Dynamic]
TIMESTAMP	

Data

Field	Comment
TEST_ID	Test identifier
NODE	Odoid
DEVICE	BLE node
ADDRESS	FM address
REALIZATION	Run identifier
PID	Packet identifier
DATA	Data value
MODE	[BLE/Trickle/FM]
NID	TX address
RSSI	
HOP	Hop count
TIMESTAMP_TX	
TIMESTAMP_RX	
TIME_BUILT	Network building time

TABLE II: MySQL database tables and fields for configuring network experiments and logging BLE mesh traffic. The *TEST_ID* field is configured as a primary key and therefore uniquely identifies and links all table entries.

collect the measurements available, while administrators may establish a secondary connection from the cloud to the testbed server and manage it.

C. Cost

The estimated cost of the testbed is shown in Table I. Its reduced cost comes from three main reasons: well known off-the-shelf components, software specifically designed using free publicly available tools and an already existing regular PC as central server. As a result, parties interested in performing BLE mesh experiments can duplicate the testbed. Alternatively, it is also possible to remotely login to it through the Fed4Fire+ project, explained next.

D. Fed4Fire+ integration

The testbed is integrated in the Fed4Fire+ project [7], the largest federation of testbeds in Europe. It is part of the European Union Horizon 2020 project, and its aim is to provide accessible and reliable experimental facilities to different research communities. A large set of tools is available to enable configuration and execution of experimental research using a wide range of testbeds. All facilities are remotely accessible through the Internet and can be used free of charge.

The BLE testbed is part of the wireless networking testbeds, and can be used to experimentally characterize any mesh protocol that runs on top of BLE. Although currently three different protocols are supported out of the box, users can flash the nRF52 BLE nodes with any custom protocol developed for them. Additionally, different network topologies can be tested and parameter sweeping experiments can be performed to optimize the performance of prototype BLE applications.

E. BLE mesh protocols

The BLE nodes are programmed with a pre-compiled binary file that contains the whole BLE stack, known as the Softdevice. Any mesh protocol running on the testbed is an application that sits on top of the Softdevice, programmed in C++ and flashed into the BLE nodes.

The standard BLE mesh protocol [6] can be used to experiment with different parameters and roles of the nodes. Apart from it, two other mesh protocols that fit within BLE operation can be selected: Trickle [13] and FruityMesh [14]. These are briefly introduced next, and the interested reader can consult a more detailed description in [15]. The justification of choosing such protocols is that both a flooding and a routing mesh flavors were desired. Trickle is a well known flooding protocol, which was implemented in the testbed prior to the release of the standard BLE mesh; and Fruitymesh is the main open source available BLE routing mesh implementation.

Trickle is a connectionless flooding protocol where BLE advertising packets are rebroadcasted until they arrive at the sink. In order to suppress excessive network activity, each node increments a counter each time a packet is overheard and proceeds to rebroadcast it only if this counter is below a certain threshold. This rebroadcast is not done immediately, but a random delay is set in order to avoid collisions. To allow a fair comparison, this threshold is set so the average number of rebroadcasts by Trickle matches the average number of hops in Fruitymesh. Note that nodes are always scanning and transmitting, which reduces end to end delay but increases energy consumption.

Fruitymesh is a connection oriented routing protocol using BLE connection packets. Clusters are formed by nodes connecting to their immediate neighbors. Then, larger clusters absorb smaller ones until the network becomes a unique fully connected cluster, resulting in a tree topology as shown in Figure 2. The protocol uses neighbor-only routing, where a packet is routed to the connected neighbor with lower number of hops to the sink. Connected nodes are synchronized: they turn on, exchange data using regular BLE connection mode and go back to sleep. This reduces energy consumption but increases end to end delay.

Finally, note that the testbed is fully BLE standard compliant, and any BLE node that is not part of the mesh network can still send packets through it. This is done by creating a GATT server on the nRF52 nodes that supports the same topic as the non-mesh node. Doing so, any node in the testbed can connect to it. Then, the connected nRF52 will parse its data, encapsulate it into a mesh data packet and inject it into the mesh network. This behavior is similar to the friend feature in the standard BLE mesh.

F. Software

Along with the three main roles of the server, all software developed for the testbed fits within three categories, described below.

1) Database logging: A MySQL 5.7 database is responsible for storing all useful data associated with a network experiment, such as parameter configuration, traffic received by every node and metadata extracted from it.

The database is divided into five tables, as shown in Table II. Both the Settings and Roles tables are configured before running a network experiment or test (used interchangeably) and become read only during its execution. The remaining tables are write only.

Settings Table

The Settings table is used to configure all the parameters for a network experiment. The TEST_ID field uniquely identifies each experiment. It is set as a primary key, becoming the field that links all tables for a specific test. Each test is repeated for a certain amount of times so the results can be averaged out to reflect statistically representative values, controlled in the field REALIZATIONS. Each of these will generate a certain number of packets that will be routed from the sources to the sinks. Note that if a single value is introduced in (SOURCE_RATE, POWER, CONINTERVAL, OUTCONNECTIONS), this value will be shared for all nodes, while introducing several will assign them individually following the order specified in the Roles table. Additionally, the SOURCE_RATE can be used for a constant or random (exponentially distributed) packet generation rate. The remaining fields are self-explanatory.

Roles Table

The Roles table specifies whether a BLE node is a source, sink or simply a mesh node that propagates packets. The NODE field identifies each Odroid, while DEVICE refers to each BLE node attached to it. The field ADDRESS reflects the software assigned address of the node within the FruityMesh network.

Nodes Table

The Nodes table is the first write only table, used for enrollment and status checking purposes. Each node initially connects to the database and identifies itself in the Nodes table. Then, it checks its role and adjusts its parameters and behavior accordingly. If the node does not have a role it means that it should not be part of the test and therefore turns off its radio. Once all parameters have been set the READY flag is set to high and the network is built. Finally, once the whole run for the test is finished (all packets have arrived at the sink) the FINISHED flag becomes active.

Control Table

The Control table provides deeper hindsight of the real-time status of the mesh network. Each BLE node periodically generates a control packet and pushes it to the database through the management network slice, with the structure introduced next. The NID field shows the address of the node, while the PID shows the packet identifier of the last data packet transmitted by it. This is used to measure the activity of the node. For its master connection (INCONN), as well as up to three of its slave connections (OUTCONN), the address of the node connected to, its average packet RSSI and the occupancy of the buffer is sent. The DEVICE_TYPE field determines whether the node is a sink, a static node or a dynamic node (an off-the-shelf mobile device that connects to the testbed). The rest of the fields are self-explanatory. Finally, note that one entry of this table will be filled per control packet received.

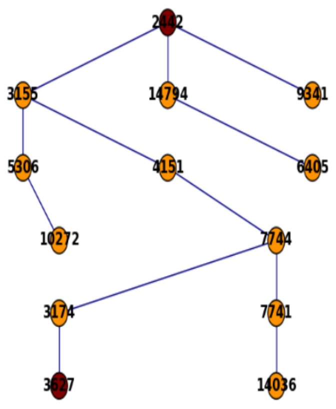


Fig. 2: Real-time network topology extracted from the control messages.

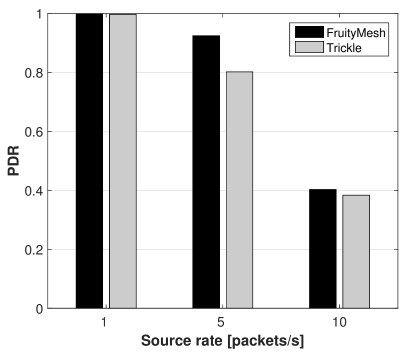
Data Table

To conclude the description of the database, the Data table stores information of the received mesh packets on every node. This is done to extract knowledge of the path, packet delivery ratio (PDR) and intermediate delay of every hop. The NID address corresponds to the node originating the packet. The HOP field determines the position of the node in the path from source to sink, and the transmitted and received timestamps (extracted from the Odroid OS clocks) are used to extract the packet delay. The TIME_BUILT field gives the network building time, i.e., from having all disconnected nodes to a fully connected single cluster. As in the Control table, each data packet received will correspond to an entry on this table.

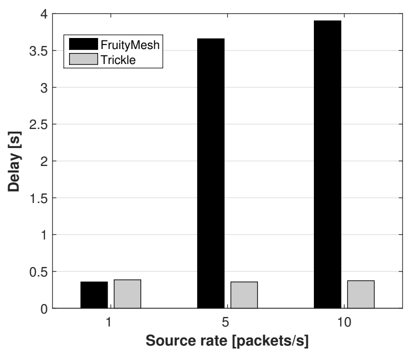
2) Network experiment management and status visualization: Three Python programs run the network experiments and provide users with real-time status of the network. The first one runs in the server and is used to configure a network test. The user fills all the necessary fields and the Settings and Roles tables are automatically configured, so no knowledge of MySQL syntax is required. A Comma Separated Values (CSV) file can also be extracted as a template for generating new experiments. Additionally, this program remotely launches the software in charge of executing the network tests.

All Odroids run this software in parallel and the server is responsible of synchronizing its execution. For the nodes attached to the 802.11 boards, the software runs directly on the server and the nodes are accessed as a virtual device.

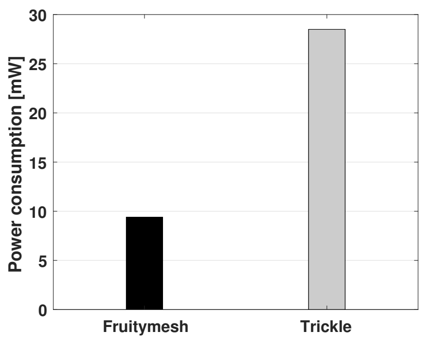
Finally, a Python front-end shows in real-time the status of the network and the network topology is extracted from the control packets, shown in Figure 2. All addresses of the nodes are shown, as well as color coded information about them: red nodes are sinks, while orange ones are regular mesh nodes.



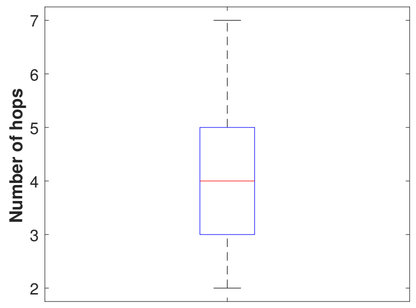
(a) Comparison metric: Packet Delivery Ratio.



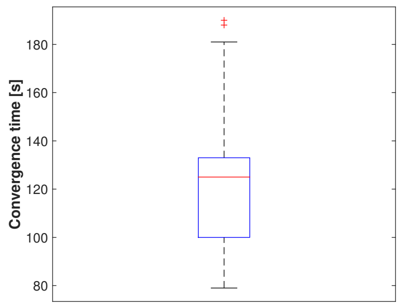
(b) Comparison metric: end to end delay.



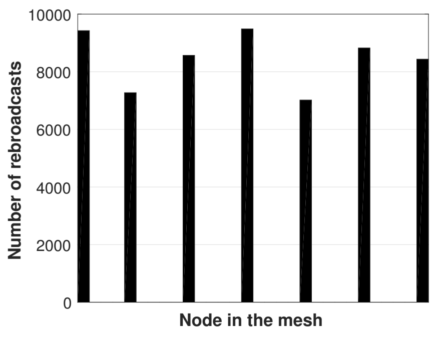
(c) Comparison metric: estimation of power consumption for successful transmission of 1 packet per interval.



(d) Fruitymesh metric: number of intermediate hops.



(e) Fruitymesh metric: network building time delay.



(f) Trickle metric: rebroadcast spread per node.

Fig. 3: Comparison of the performance of FruityMesh and Trickle as BLE mesh protocols.

3) Gateway and cloud: The server acts as a gateway to allow internet connectivity and remote access to the Odroids. Additionally, it runs an NTP server for clock synchronization.

Control and data messages parsed and received at the testbed server are pushed to the cloud server using MQTT, a TCP/IP lightweight publish-subscribe messaging protocol. Doing so, any user can set up an MQTT broker and connect to the cloud server to receive all testbed data in real time.

G. *Limitations*

The proposed framework has three main limitations: lack of specific co-channel interference mitigating features for the traffic generated by the 802.11 boards (apart from native 802.11 CSMA/CA and BLE AFH); limited number of mesh protocols implemented and finally a moderate number of mesh nodes to perform network characterization. Given the modular design approach chosen the latter can be easily solved, while the former two are left as future work.

III. Case Study

This section introduces the results of a measurement driven comparison of Trickle [13] and Fruitymesh [14] using the testbed. An overview of them is given, but the interested reader is encouraged to consult [15] where they were originally published in further detail, along with a more extensive explanation of the scenario and metrics used.

A. *Scenario*

The experiments conducted in [15] were performed at an early stage in the development of the testbed, with a smaller network size than the current one. A total of nine nodes where deployed: one acting as a source, another one as a sink and the remaining seven as mesh nodes, forwarding traffic to the sink.

Trickle and Fruitymesh test realizations are alternated. For each protocol, a total of 256 packets per realization are sent by the source with constant rates of 1, 5 and 10 packets per second, and each source rate is repeated for 20 realizations in order to average the results. This accounts for 120 network realizations and 30720 packets per protocol in total, where the transmission power is set to -20 dBm, the interval duration to 100ms and the buffer size to 15 packets.

The two mesh protocols are compared in terms of PDR and end to end delay (both measured at the sink), and additionally an estimation of the power consumption for transmitting a packet in each interval. This is based on the nRF52 online power profiler [8], developed by Nordic Semiconductor based on actual measurements on the nodes. Estimates of the average consumption of scanning and transmitting events are given, as well as startup, packet processing and idle operations. For each mesh protocol, we account for all events present in the interval and estimate the total consumption. For Trickle, we assume that the node rebroadcasts one packet during an interval and is scanning for its whole duration. For Fruitymesh, we assume that the node has one master and three slave connections (the usual values in the testbed). The node wakes up, exchanges one data packet with the next hop plus an additional dummy packet with the rest of its neighbors to maintain the connections and then goes to sleep for the rest of the interval.

Finally, since local information of every node can be consulted, protocol specific metrics are obtained. For Trickle, the total number of rebroadcasts per node is presented. For Fruitymesh, the average number of hops from source to sink and the time needed to build the fully connected network are given.

B. Results

Figure 3 shows the results obtained from the measurement campaign. In terms of PDR both protocols show a decreasing performance with the source rate: Trickle does not implement acknowledgements and packets are lost due to interference and collisions; FruityMesh retransmits the packet until it is acknowledged, which leads to buffer overflow that cannot keep up with a constant source rate.

The delay of Trickle is the lowest possible since packets are flooded and may travel all paths in the network, including the shortest one. For FruityMesh, the delay suffers from a saturation effect since packets travel through all queues of all intermediate nodes in the path.

In terms of power consumption, FruityMesh outperforms Trickle. This is due to the synchronized nature of the connected mesh where nodes can sleep, contrary to Trickle where nodes listen for the whole interval.

As a conclusion, if the throughput in the network is constant the performance in terms of PDR of the two mesh protocols can be comparable, while Fruitymesh can reduce its power consumption at the cost of increased end to end delay when compared to Trickle. Therefore, a connected BLE mesh may be more advantageous than a flooding mesh depending on the target application. This result needs to be taken into account for future versions of the current BLE mesh protocol.

Regarding the Fruitymesh specific metrics, it can be seen that the time needed to have a fully connected network and the average amount of hops in the path can be quite considerable. This is due to the clustering process present in Fruitymesh, where nodes connect at random to each other. This procedure is repeated each time a test realization is performed, since the whole network is reset. Given its random nature, some network realizations may have excessive long paths and excessive time may be needed to converge into a single cluster network.

In the case of Trickle, the rebroadcast spread per node is fair, with no node showing a particularly different activity. This fairness is a result of the random delay between overhearing a packet and rebroadcasting it: nodes that have a shorter delay will in fact rebroadcast it, while nodes with a longer delay will be able to overhear all previous nodes and their packet counter will exceed the threshold, thus suppressing the transmission. Regarding this threshold, as it is set to be equal to the average number of hops in the network, each node rebroadcasts an average of 30720/4=7680 packets.

IV. Conclusions

This paper presents the design and implementation choices of a BLE testbed used to experimentally characterize mesh protocols. The testbed is low-cost, easily expandable due to its modular design, supports multiple mesh protocols and is fully automated. Additionally, it offers the possibility of establishing a remote connection to it and perform any experiment to third parties that may be interested in doing so. All hardware, software and architecture of the testbed is described in detail to inspire researchers that need to build their own testbed. Next, a comparison of existing mesh protocols that fit within BLE operation using the testbed is also given.

For the future we expect that this testbed will help optimize the performance of future releases of BLE mesh, as well as serve as a powerful tool for research in wireless mesh network topics such as network tomography or end to end quality of service.

ACKNOWLEDGMENTS

This work is supported by IWT CELTIC O&O project ASUA/RoCCS. Yuri Murillo is funded by a PhD fellowship of the Research Foundation Flanders (FWO).

References

[1] Cisco, "Cisco visual networking index: Global mobile data traffic forecast update, 2016–2021," *White Paper*, March 2017.

[2] H. Peng, Y. Xiao, Y.-N. Ruyue, and Y. Yifei, "Ultra dense network: Challenges, enabling technologies and new trends," *China Communications*, vol. 13, pp. 30–40, 02 2016.

[3] Qualcomm, "Csrmesh product page," https://www.qualcomm.com/solutions/networking/features/csr-mesh, accessed: 2018-02-01.

[4] L. Leonardi, G. Patti, and L. L. Bello, "Multi-hop real-time communications over bluetooth low energy industrial wireless mesh networks," *IEEE Access*, pp. 1–1, 2018.

[5] H. S. Kim, J. Lee, and J. W. Jang, "Blemesh: A wireless mesh network protocol for bluetooth low energy devices," in *2015 3rd International Conference on Future Internet of Things and Cloud*, Aug 2015, pp. 558–563.

[6] Bluetooth SIG, "Ble mesh profile specification v1.0," https://www.bluetooth.com/specifications/mesh-specifications, accessed: 2018-02-01.

[7] "Fed4Fire european project page," https://www.fed4fire.eu/, accessed: 2018-02-01.

[8] Nordic Semiconductor, "Nordic nrf52 development board website," https://www.nordicsemi.com/eng/Products/Bluetooth-low-energy/nRF52-DK, accessed: 2018-02-01.

[9] Hardkernel, "Odroid-C2 single-board computer website," http://www.hardkernel.com/main/products/prdt_info.php?g_code=G145457216438, accessed: 2018-02-01.

[10] Mikrotik, "RB912UAG-2HPnD routerboard website," https://mikrotik.com/product/RB912UAG-2HPnD, accessed: 2018-02-01.

[11] Open-mesh, "B.A.T.M.A.N. advanced project page," https://www.open-mesh.org/projects/batman-adv, accessed: 2018-02-01.

[12] "ser2net sourceforge project page," https://sourceforge.net/projects/ser2net/, accessed: 2018-02-01.

[13] P. Levis, T. Clausen, J. Hui, O. Gnawali, and J. Ko, "The trickle algorithm," Internet Requests for Comments, RFC 6206, March 2011. [Online]. Available: https://tools.ietf.org/html/rfc6206

[14] M-Way Solutions, "Fruitymesh project page," https://github.com/mwaylabs/fruitymesh/wiki, accessed: 2018-02-01.

[15] Y. Murillo, B. Reynders, A. Chiumento, S. Malik, P. Crombez, and S. Pollin, "Bluetooth now or low energy: Should ble mesh become a flooding or connection oriented network?" in *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC) - Special Session SP-04 on Resource-Efficient, Reliable and Secure Internet of Things in the 5G Era*, Oct 2017.

Yuri Murillo (yuri.murillo@esat.kuleuven.be) obtained his MSc degree in telecom engineering from Public University of Navarre (UPNA), Spain, in 2013. He wrote his master thesis at TU Delft, the Netherlands. From 2013 to 2015 he worked as an IT security consultant at GMV and obtained a MSc in business administration from UAH, Madrid. He is currently pursuing a PhD at KU Leuven. His main research interests include optimization of wireless mesh networks and channel prediction.

Brecht Reynders (brecht.reynders@esat.kuleuven.be) obtained his BSc and MSc degree in electrical engineering from KU Leuven, Belgium, respectively in 2012 and 2014. Currently he is working towards his PhD at KU Leuven, focusing on multi-technology routing. In 2016 he obtained the Belgian FITCE Young ICT Personality of the Year national award. His main research interests are long range communication, distributed computing and network modeling.

Alessandro Chiumento (alessandro.chiumento@esat.kuleuven.be) received his PhD degree in cellular network management from imec, Leuven, Belgium, in 2015. He is currently with the Department of Electrical Engineering, KU Leuven. His research interests include massive machine-to-machine communication, channel prediction, very dense networks, and the application of machine learning to theoretical problems in telecommunication and information management.

Sofie Pollin (sofie.pollin@esat.kuleuven.be) obtained her PhD at KU Leuven in 2006. She continued her research on wireless communications at the University of Berkeley, California. In 2008 she returned to imec to become a scientist in the green radio team. Since 2012, she has been an assistant professor in the Electrical Engineering Department at KU Leuven. Her research centers around networked systems that require networks that are ever more dense, heterogeneous, battery powered, and spectrum constrained.