BLUES: A self-organizing BLE mesh-network paradigm for IoT environments

Emanuele Giacomini, Francesco D'Alterio, Andrea Lacava, Francesca Cuomo
Sapienza University of Rome, 00184 Rome, Italy
Email: {giacomini.1743995, lacava.1663286}@studenti.uniroma1.it
Email: {francesco.dalterio, francesca.cuomo}@uniroma1.it

***Abstract*—The introduction of new key features into the core specification of Bluetooth Low Energy (BLE) increased its potentialities, thus paving the way to the development of new networking paradigms. The ability for a single node to participate in more than a piconet and to assume both the role of master and slave makes it possible the formation of multi-hop networks that can be used in several emerging application scenarios.**

Additionally, the inherent low power consumption at the cost of contained throughput makes this technology appealing for Internet of Things (IoT), where power memory and capacity constrained devices exchange messages at relatively low data rates.

In this paper, we devise a two layers BLE mesh-based networking paradigm obtained by generalizing Android BE-MESH for hardware-independent sensor networks. Each node enforces a plug-and-play architecture which makes it able to autonomously switch between client and server role, discover and connect to existing scatternets and relay messages through them, making the network able to extend and self re-organize in a distributed fashion.

To have our implementation ready for IoT systems we based it on the ESP32 off-the-shelf board. We describe both the implemented functions as well as some practical results proving the effectiveness of the framework in some tested configurations.

***Index Terms*—IoT, Bluetooth Low Energy, Mesh networks, Ad-hoc networks.**

I. Introduction

The last few years have been witnessing the birth and development of Bluetooth Low Energy (BLE), a brand new communication technology designed for energy-constrained systems which guarantees theoretical bit-rates of 1 Mbps and 2 Mbps for BLE 4.2 and BLE 5, respectively [1].

Whilst this technology is actually mainly deployed to create point-to-point links or star networks, where peripherals are connected to a central device, it is possible to exploit default functionalities to achieve more complex network topologies. By exploiting the Generic Access Profile (GAP) and the Generic Attribute Profile (GATT) services of BLE it is possible to distinguish several classes of devices to interact with each other and to allow multi-hop network formation and routing.

Due to the infrastructure-less nature of this technology, small power requirements and widespread usage, it represents a cost-effective solution to be embedded in Internet of Things (IoT) platforms generally being tiny devices with tight constraints with regard to batteries, available memory and computational resources. BLE in a mesh setting can be used in a multiplicity of scenarios, like IoT, edge computing networks, infrastructure-less networks or to support existing infrastructures to overcome their own limitations (e.g. in terms of coverage).

Even though BLE supports only very short-range communications in line-of-sight, thereby reducing the effective achievable distance to tens of meters, it could leverage on a huge amount of saved energy allowing the battery of the devices to last for a very long time period. These savings, however, can be further improved by considering better mesh communication protocols than those suggested by the standard, which is still based on techniques that do not consider the energy limit of IoT devices, using for instance flooding to deliver messages to the entire network.

In this context, we present BLUES (BLUetooth ESp32 mesh), a self-organizing BLE Mesh network designed for IoT, obtained by generalizing Android BE-MESH [2] for hardware-independent sensor networks. We show as a proof of concept an implementation on Espressif Systems product ESP32, a common Arduino-like board which is open source and implemented in most of sensor IoT environments [3].

The rest of the paper is organized as follow. In Section II, we discuss the related work discussing the need of developing new paradigms for BLE, while in Section III we illustrate our system design. In Section IV we present our testbed and implementation for ESP32, together with some preliminary tests conducted on this platform. Finally, Sections V and VI conclude our work by showing the possible use cases of BLUES in some well known IoT fields and by proposing some future scenarios that could be investigated.

II. Related Work

Due to its low power, bandwidth usage (ISM) and data rates, BLE technology has been gaining ground in a wide range of fields requiring short-range communications, among these smart homes, industry 4.0 and health monitoring to cite a few. A brief tutorial of the main procedures to establish BLE connections is reported in [4] where also solutions to solve some contention issues in dense IoT scenarios are presented. Groundbreaking smart-home solutions are presented in [5], [6] and [7], which discuss the interaction between smart sensors and, respectively, android systems, centralized management servers and individual BLE devices, in point-to-point networks.

978-1-7281-7374-0/20/$31.00 ©2020 IEEE
DOI 10.1109/WoWMoM49955.2020.00075

Smart manufacturing is investigated in [8] and [9], both proposing BLE monitoring technologies for industrial plants. Scenarios leveraging BLE for ad-hoc body networks are presented in [10], [11] and [12], proposing three alternatives based on star topology to collect and share data with backhaul cloud infrastructure.

Despite the fact that BLE paradigm has been investigated from various perspectives, its application in the development of mesh networks is still an open field. Some work present in-depth analysis on several aspects related to BLE networking: work in [13] investigates the probability of node isolation and K-connectivity for BLE-meshes while [14] presents an extensive analysis regarding the inference between network parameters. Other works, conversely, propose practical implementations in the shape of protocols or frameworks but with some remarkable differences with our approach. In [15] a protocol allowing real-time communication for BLE mesh networks is proposed. Differently from BLUES, this protocol envisions a master/slave paradigm where several slave nodes communicates only with a single master using a TMDA approach, and also considers a static scenario with static routing performed offline. In [16] the authors present an Android library allowing public safety communications over the same type of networks. This dynamic approach leverages both the Received Signal Strength Indicator (RSSI) and hop count for selecting the best path among a small network. While representing a valuable tool for computer and smartphone networks it suffers from high computational overhead, hampering its usage in IoT scenarios. Finally, other contributions are provided by [17], proposing a testbed for IPv6 over Low power Wireless Personal Area Networks (6LoWPAN) technology for BLE networks and by [18] which discusses a cluster-based on-demand routing protocol implementing a flooding avoidance mechanism that reduces routing discovery broadcast messages. The BLUES solution here presented instead is the first tentative to build, on top of an open source and versatile System-On-a-Chip, a mesh network and an efficient routing mechanism for IoT smart devices based on BLE.

III. System design

Every BLUES node is composed of two main layers: the *kernel layer* and the *routing layer*. The kernel level is intended to wrap and hide all the hardware functionalities and provide a unified interface to the upper level, which is where BLUES logic is actually implemented, making the routing paradigm hardware independent. At the routing level we define two roles that a single device can assume during the network lifetime: the `server` and the `client`.

The server is responsible of message dispatching inside the network and bases its functionalities on a single BLE service which, in turn, leverages a single BLE *characteristic*. Moreover, a server is the manager of its own piconet, formed of its set of clients, and also communicates with other servers in order to create complex scatternets.

To avoid any loss of performance, the BLE standard recommends a maximum number of active connections per BLE Service. To this aim, we designed every BLUES server to have maximum 7 contemporary connections being freely divided into `server2server` and `server2client` ones. By default we allow a maximum of 4 incoming connections for clients and 3 outgoing connections for servers, but these numbers can be adapted in order to create specific network topologies.

On the other hand, the `client` represents a terminal node connected to a single server and has only the ability to send and receive messages within the network. This role has been kept as simple as possible in order to reduce the computational overhead, making client nodes more suitable to perform additional services.

A. *Basic BLUES node behaviour*

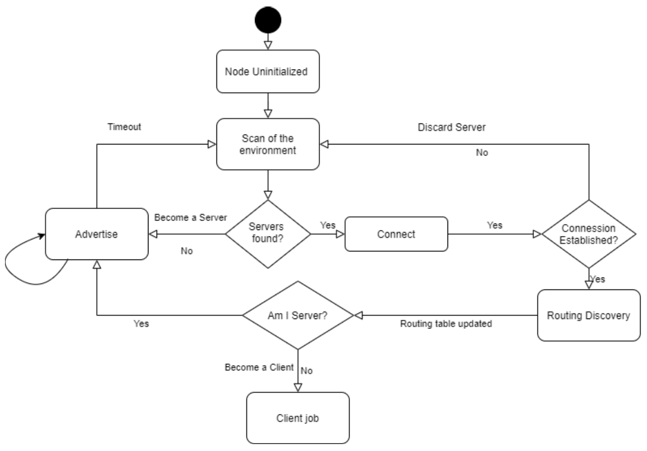


Fig. 1: Basic BLUES node behaviour.

The basic behaviour of any BLUES node is to assume a role as soon as it becomes operative. This functionality allows a complete self-configuration of the network without the need for external intervention. It is based on a Finite State Machine (FSM) and can be easily adapted to any need by forcing the initial state assumed by the FSM and thus allowing a node to start directly as a client or as a server. If the device starts from an undefined state, as a first action it searches for other BLUES active servers in the surrounding environment by performing a BLE Scan. The result of this action is a list of relatively close active servers.Given the mesh nature of the network and to allow better communication and reduced latency, this list is sorted according to the best RSSI through a classic insertion sort in-place algorithm. This algorithm has been chosen since it does not add any spatial cost on the device memory as well as a linear temporal cost, except for worst case scenarios. Considering that this list is almost completely sorted any time, due to the fact that a new device is listed in the correct order, as long as it has been discovered (except when a device change its position), worst case is very unlikely to happen.

Since every found server can provide connectivity to a limited number of devices, the node has to query these servers until an available spot for a client-server connection is found or a timer expires. In case of success, the server establishes a connection with the device and subsequently launches a *RoutingDiscovery* routine to notify all the other devices that a new element joined the network. This routine is described in detail in the next section.

Conversey,If at the end of the scanning there are no servers or no availability to establish a connection, the new device assumes the role of server, connects to the nearest server to receive the routing table and starts advertising the possibility to accept new clients. After an arbitrary period of advertisement, that is completely variable and adaptable to any possible use case a server re-enters the scanning phase and if it finds new servers it connects to the latter in order to merge the two independent scatternets into a larger one. A visual representation of the BLUES node behaviour is shown in Figure 1.

B. BLUES routing table

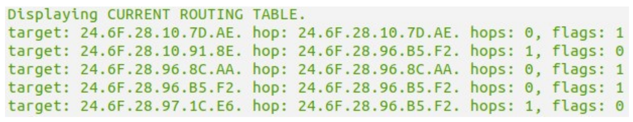
Every node has its own routing table which is composed of several records, one per connected device excluding the local host. Each record consists of the address of the reachable device, the address of the node working as gateway towards that device (next hop) and the number of hops required to reach an intended destination.

Considering that each device is identified by its Bluetooth Device Address (BDA), we define a single entry of the routing table as a tuple with the following format:

$$(BDA_{target}, BDA_{nexthop}, num_{hop}, flags).$$

The size of a BDA is 6 bytes while the sizes of the flags and of the number of hops field are 1 byte each. Therefore any single entry requires 14 bytes and the whole routing table occupies $14 \times n$ bytes with n equal to the number of network devices. Figure 2 represents an actual routing table generated by BLUES.

While in some Android BLE systems the BDA is generated in a pseudo random way [19], then inducing developers to implement custom identification paradigms [2], in BLUES we use a static value to address a device in order to have the same address at every run. This behaviour however was designed for convenience only and does not have any impact on the performance, nor on the operation of the network.



```
Displaying CURRENT ROUTING TABLE.
target: 24.6F.28.10.7D.AE. hop: 24.6F.28.10.7D.AE. hops: 0, flags: 1
target: 24.6F.28.10.91.8E. hop: 24.6F.28.96.B5.F2. hops: 1, flags: 0
target: 24.6F.28.96.8C.AA. hop: 24.6F.28.96.8C.AA. hops: 0, flags: 1
target: 24.6F.28.96.B5.F2. hop: 24.6F.28.96.B5.F2. hops: 0, flags: 1
target: 24.6F.28.97.1C.E6. hop: 24.6F.28.96.B5.F2. hops: 1, flags: 0
```

Fig. 2: Example of BLUES Routing Table.

C. BLUES routing protocol

As indicated in Fig. 1 every time two nodes establish a connection (being either a client-server or a server-server one) they exchange their routing tables by performing a *RoutingDiscovery*. This is used by a server to inform clients of the available routes or to exchange its routing table with the one of another server, thus extending the knowlegde of the network. In general two BLUES nodes, *deviceA* and *deviceB*, that establish a connection for the first time run the *RoutingDiscovery* routine that consists of three main steps:

- *deviceA* sends a *DiscoveryRequest* to *deviceB*. The *DiscoveryRequest* message contains an empty payload used just to notify the beginning of the routine;
- *deviceB* replays with a *DiscoveryResponse* message to *deviceA*, containing its own routing table as payload;
- if *deviceA* also has its own routing table, then sends it back to *deviceB* with a *DiscoveryResponse*.

Whenever a server gets a *DiscoveryResponse*, it updates its routing table. It also inserts any update into another table, called *RCT*, which keeps track of recent changes in the discovered routes. The server then transmits it to its neighbours, instead of the complete routing table, thus obtaining a reduction of the injected network traffic, especially in cases of large networks.

Each *RCT* entry is labeled with a *state* representing which action was performed on it: (i) entry added, indicating that it has just been inserted into the routing table; (ii) entry updated, if a better path to that device has been discovered; (iii) entry removed, to notify a device which is not part of the scatternet anymore.

Any node which receives a *RoutingUpdate* message may choose whether to accept or discard updates, depending on the current state of its routing table. For instance, if the *deviceA* already knows a route to reach *deviceC* and it receives an update coming from *deviceB* containing a higher cost path for the very same destinations, it just drops it.

IV. IoT Testbed

To test the whole implementation of our BLUES solution we used some low-power ESP32 modules, general-purpose IoT chips with integrated BLE adapters, where each device is fully autonomous and able to execute the whole BLUES stack. The source code is available in [20]. It is currently implemented to be compatible with the aforementioned hardware, but it can be easily ported to any required hardware platform re-implementing the *Kernel* module. Furthermore, we advise to use this code as a routing middle-layer, thus shadowing any routing related aspect while building an independent application layer on top of it.

In the following subsections we give a quick overview of the ESP32 Platform, its features and the reasons why we selected it for our work. Then we discuss the current implementation providing several numerical results so as to prove the effectiveness of our mechanism.

A. ESP32 Platform

To develop a testbed for BLUES we selected the open source `ESP32-WROOM-32` System-On-a-Chip (SoC) due to high affinity with the ideal concept of IoT smart device: a versatile SoC able to achieve interconnection, on medium-range, using both BLE or WiFi, while presenting the key characteristics of small size, memory, power consumption and

CPU usage. More specifically, this board presents a 4.2 BLE chip, a WiFi 802.11 b/g/n chip, 520 kB of RAM, 4 MB of flash memory and two low-power 32-bit LX6 microprocessors, it is also open-source and Arduino compatible.

The firmware provided for this platform enforces the specifications defined in section III for both *kernel layer* and *routing layer*. The kernel layer manages all the interactions with the BLE stack implemented by the Bluedroid module inside the board. Moreover, the kernel layer handles the GATT events by splitting *Server* operations (i.e. accepting new connections, notifications etc.) handled with *GATT-S handler* and *Client* operations (i.e. sending connection requests, read/write operations) with the module *GATT-C handler*.

GAP operations (i.e. scan phase, advertising, gathering data from surrounding nodes) are in the responsibility of the respective *GAP handler* module. In order to handle all these events at a higher level, the kernel module allows the subscription of callback functions to these specific events. This results in the assumption of different behaviours for the node according to its current state, since it establishes both which module and also which way it maps every kernel event to a specific action. Indeed, it is thanks to this facility that the *routing layer*, which manages the formation of a network together with its high level operations (such as handling of roles, message exchange and parsing), can manage the node behaviour as already shown in Figure 1.

B. Experimental Results

The software stack introduced above has been installed on several ESP32 devices so as to investigate the evolution of the network. One particular aspect we want to investigate is the *convergence time*, which can be expressed as the interval of time that occurs from the startup of the first device to the full formation of the network, intended as the status where each node is able to communicate with any other node through a path defined in its routing table. We performed several experiments in this regard leveraging up to 7 devices capable of autonomously scanning the surrounding environment and decide which role to assume (Fig. 3). Values of the used parameters are given in Table I.

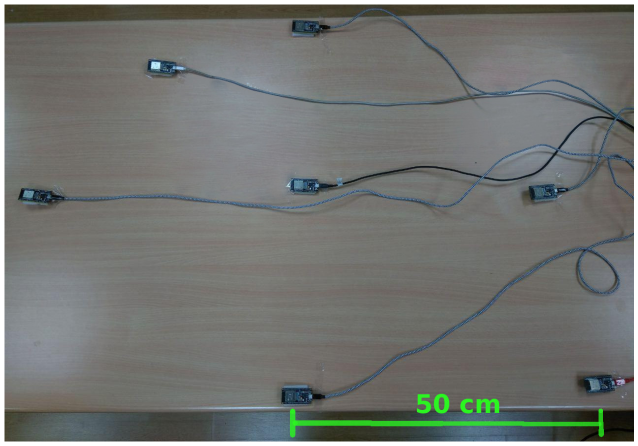


Fig. 3: Testbed composed by 7 ESP32 boards

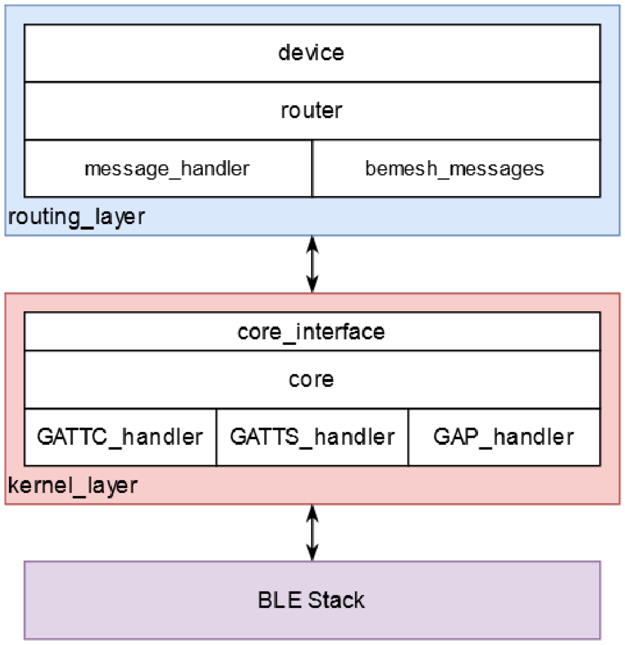


Fig. 4: BLUES stack.

TABLE I: Experiment Parameters.

Parameter	Value
max incoming connections	2
max outgoing connections	2
BLE connection timeout	$5s$
duration of scanning operation	$1-10s$

In order to present a quite complex scenario, characterized by different kind of topologies, the number of incoming connection per device has been set to 2, thus limiting the maximum number of client devices per server. In normal use cases this value can be set closer to the hardware specific limitations. Moreover, all the nodes have been powered simultaneously, in order to avoid any measurement bias.

In Figure 5 we show the average time required to achieve convergence as a function of the network size. It is possible to notice a substantial increase in the required time passing from 3 to 4 devices, caused by the the introduction of a new server in the topology increasing its complexity and therefore requiring more effort to reach the convergence. This phenomenon also occurs in case of a higher number of devices but in a lesser extent. This is mostly due the higher degree of freedom presented by the network in this case which allows the coexistence of several network topologies considering the same number of devices.

This latter claim is motivated in Fig. 6 which presents the number of exchanged routing updates related to the number of interconnected devices. While networks with a small number

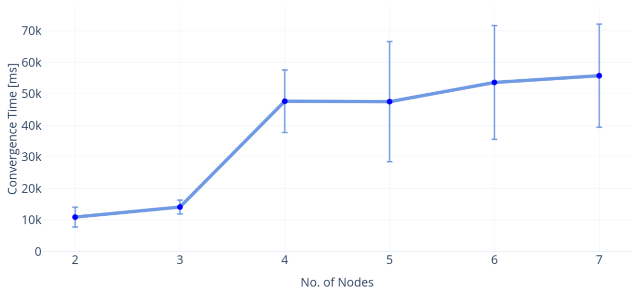


Fig. 5: Average time to reach convergence as a function of network nodes. 95% confidence intervals are shown.

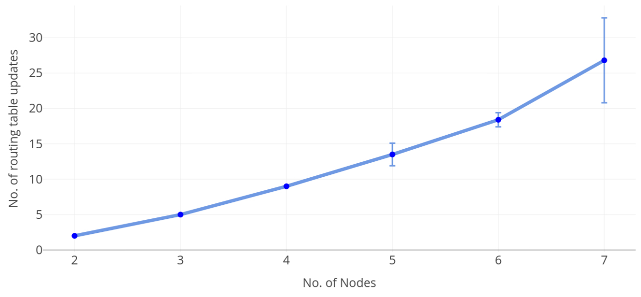


Fig. 6: Average number of routing updates required to reach convergence as a function of network nodes. 95% confidence intervals are shown.

of devices, around 3 or 4 units, require a fixed number of updates to reach convergence, this number starts to experience a high variability when the number of devices grows, hence suggesting the formation of diverse network configurations for a given number of devices.

Results obtained so far could provide an overview of the effectiveness of our framework comparing its cost with the *flooding* paradigm's one, that is the option advised by the Bluetooth SIG [1] to develop BLE meshes. Interpolating data in Fig. 6 it is possible to generalize the provided results for wider networks: using MATLAB software suite we obtained the following second degree polynomial function indicating the expected number of exchanged routing updates for a network of N nodes

$$convergence_{BLUES}(N) = 0.5327 \times N^2 + 0.0577 \times N - 0.1875 \; .$$

In case of a network with N devices trying to send M messages, assuming, without loss of generality, that all the possible network configuration are uniformly distributed,any message should traverse on the average $\frac{N}{2}$ devices before reaching the destination, thus obtaining the following number of packet transmissions.

$$\begin{aligned} & sendMessages_{BLUES}(N, M) \\ & \quad = (0.5327 \times N^2 + 0.0577 \times N - 0.1875) + (N/2) \times M \end{aligned}$$

On the other hand, flooding paradigm doesn't perform any network topology discovery routine, hence experiencing a huge number of retrasmissions: considering its most general form where each node is required to retransmit in broadcast any message of which it is not the intended receiver, the aforementioned network requires this number of transmissions

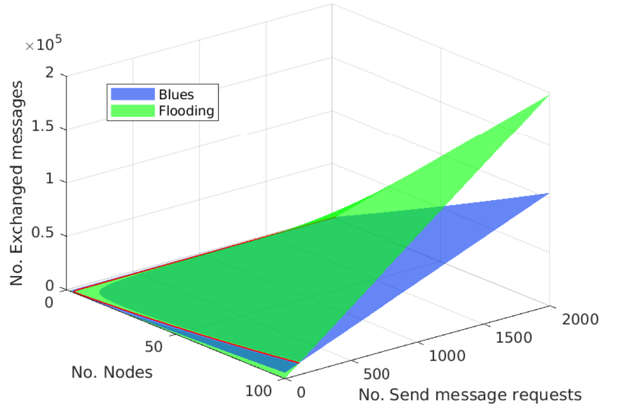


Fig. 7: Comparison between BLUES and flooding in terms of packet transmissions for number of network nodes and send-message requests

$$sendMessages_{FLOODING}(N, M) = M \times (N - 1) \; .$$

Figure 7 presents a comparison between BLUES and flooding showing the higher efficiency of this latter in case of small networks, due to its simple infrastructure-less nature. Conversely, BLUES performs better when the number of sent messages increases (so when the network grows), nearly halving the cost in most of the cases.

V. Possible Use-cases and future Work

The proposed paradigm presents high versatility and high decoupling to the underlying hardware infrastructure making the possible use cases manifold, in particular we advise the following ones:

- **Edge Computing:** cases where a sensor network infrastructure, supported by a relatively far cloud performing the whole computation, is not effective. This can be the case of massive networks, where the huge amount of shared data could not be sent over the backhaul network but must be partially processed locally in order to avoid saturation. BLUES may represent an light mesh infrastructure that can be used to perform some distributed processing in accordance to the edge computing paradigm.
- **Faulty-backhaul tolerant networks:** situations where a small-area network must be deployed in case of a fluctuating back haul infrastructure, therefore requiring to the network to autonomously reconfigure in order to share and manage data which will be eventually collected at later time. Emergency networks represent the typical

example of this use case, where the backhaul infrastructure becomes unreliable or even inaccessible for some time.

- **Infrastructure-less IoT networks:** obtained taking the previous case to the extreme. Here a small sensor network must perform in isolation and final data will be collected over scheduled periods of time. It is a valuable candidate for environmental monitoring systems, which result very complex to be reached by supporting infrastructure, like volcanoes, mountain ranges or sea depths.
- **Supporting existing infrastructures:** it could provide support to already existing wireless technologies in order to increase performance or overcome inherent limitations of the former. A prime example is the possible synergy with low-power wide-area networks like LoRaWAN, increasing its granularity considering a small network enforcing our paradigm in place of a single end node
- **IoT Domotic:** One of the greatest risk for IoT domotics is the exposure of multiple agents to the internal and external Internet. In this case BLUES could expose a single node to the external network acting as a gateway while hiding the internal nodes by applying known techniques of swarm robotics. Moreover, through the implementation of the 6LoWPAN over BLE standard it would be possible to completely hide our paradigm avoiding well-known problems of compatibility between different IoT devices.

VI. Conclusions

This paper presents an innovative autonomous networking framework with self-reconfiguring capabilities for next-generation massive IoT. The technology hereby proposed aims at boosting the potentials of IoT networks, addressing common limitations affecting widespread networking paradigms for IoT, such as fixed topology requirements or limited number of nodes by introducing multi-hop interconnection between resources-constrained devices leveraging Bluetooth Low Energy 4.2+ protocol stack functionalities. More specifically, the devised approach uses GATT and GAP specifications to define two classes of devices, *client* and *server*, and their mutual interaction. Each node autonomously assumes one of the two aforementioned roles and establishes connections following a greedy approach relying on network specific parameters only, with no need for any external aid.

Several experiments have showed the ability for the network to reach convergence over a finite time span, requiring a small number of exchanged routing updates, hence making it a valid alternative to the widely used flooding algorithm in situations of massive message exchange.

Future work will focus on providing optimal parameters configurations for predefined classes of services in order to optimally reduce convergence times and number of exchanged messages. We also aim at providing compatibility to well-known routing algorithms in order to facilitate the exploitation of their features on ad-hoc IoT networks.

Acknowledgments

The authors would like to thank Andrea Del Vecchio and Riccardo Caprari for their support in the ESP32 software design.

References

[1] B. SIG, “Bluetooth specification version 5.2,” ”https://www.bluetooth.org/docman/handlers/downloaddoc.ashx?doc_id=478726”, p. 187, 2020, [Online; accessed 17-February-2020].

[2] A. Lacava, G. Nero, P. Locatelli, F. Cuomo, and T. Melodia, “Demo abstract: Be-mesh: Bluetooth low energy mesh networking,” in *IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*, April 2019, pp. 989–990.

[3] A. Maier, A. Sharp, and Y. Vagapov, “Comparative analysis and practical implementation of the esp32 microcontroller module for the internet of things,” in *2017 Internet Technologies and Applications (ITA)*, Sep. 2017, pp. 143–148.

[4] A. F. Harris III, V. Khanna, G. Tuncay, R. Want, and R. Kravets, “Bluetooth low energy in dense iot environments,” *IEEE Communications Magazine*, vol. 54, no. 12, pp. 30–36, December 2016.

[5] M. Porjazoski, P. Latkoski, and B. Popovski, “Bluetooth low energy-based smart home android solution,” in *IEEE EUROCON 2019 -18th International Conference on Smart Technologies*, July 2019, pp. 1–5.

[6] M. Collotta and G. Pau, “A novel energy management approach for smart homes using bluetooth low energy,” *IEEE Journal on Selected Areas in Communications*, vol. 33, no. 12, pp. 2988–2996, Dec 2015.

[7] H. Kondo and K. Takami, “Individualized electric power management system using ble tag at a shared house,” in *Int. Conference on Information Networking (ICOIN)*, Jan 2018, pp. 86–91.

[8] R. Tei, H. Yamazawa, and T. Shimizu, “Ble power consumption estimation and its applications to smart manufacturing,” in *54th Annual Conference of the Society of Instrument and Control Engineers of Japan (SICE)*, July 2015, pp. 148–153.

[9] R. N. Gore, H. Kour, M. Gandhi, D. Tandur, and A. Varghese, “Bluetooth based sensor monitoring in industrial iot plants,” in *Int. Conference on Data Science and Communication (IconDSC)*, March 2019, pp. 1–6.

[10] T. M. Siham Sayeed, M. T. Rayhan, and S. Chowdhury, “Bluetooth low energy(ble)based portable medical sensor kit platform with cloud connectivity,” in *Int. Conference on Computer, Communication, Chemical, Material and Electronic Engineering*, Feb 2018, pp. 1–4.

[11] M. Donati, A. Celli, A. Ruiu, S. Saponara, and L. Fanucci, “A telemedicine service platform exploiting bt/ble wearable sensors for remote monitoring of chronic patients,” in *7th Int. Conf. on Modern Circuits and Systems Technologies (MOCAST)*, May 2018, pp. 1–4.

[12] P. Paladugu, A. Hernandez, K. Gross, Y. Su, A. Neseli, S. Gombatto, K. Moon, and Y. Ozturk, “A sensor cluster to monitor body kinematics,” in *IEEE 13th Int. Conference on Wearable and Implantable Body Sensor Networks (BSN)*, June 2016, pp. 212–217.

[13] S. M. Darroudi and C. Gomez, “Modeling the connectivity of data-channel-based bluetooth low energy mesh networks,” *IEEE Communications Letters*, vol. 22, no. 10, pp. 2124–2127, Oct 2018.

[14] A. Chiumento, B. Reynders, Y. Murillo, and S. Pollin, “Building a connected ble mesh: A network inference study,” in *IEEE Wireless Communications and Networking Conference Workshops (WCNCW)*, April 2018, pp. 296–301.

[15] L. Leonardi, G. Patti, and L. Lo Bello, “Multi-hop real-time communications over bluetooth low energy industrial wireless mesh networks,” *IEEE Access*, vol. 6, pp. 26 505–26 519, 2018.

[16] B. Zhang, Y. Wang, L. Wei, Q. Jin, and A. V. Vasilakos, “Ble mesh: a practical mesh networking development framework for public safety communications,” *Tsinghua Science and Technology*, vol. 23, no. 3, pp. 333–346, June 2018.

[17] H. Kang, C. Kim, and S. Koh, “Iso/ieee 11073-based healthcare services over iot platform using 6lowpan and ble: Architecture and experimentation,” in *International Conference on Networking and Network Applications (NaNA)*, July 2016, pp. 313–318.

[18] C. Jung, K. Kim, J. Seo, B. N. Silva, and K. Han, “Topology configuration and multihop routing protocol for bluetooth low energy networks,” *IEEE Access*, vol. 5, pp. 9587–9598, 2017.

[19] ”https://blog.bluetooth.com/bluetooth-technology-protecting-your-privacy”, [Online; accessed 23-January-2019].

[20] ”https://github.com/BE-Mesh/BLUES-esp32”, [Online; accessed 14-March-2020].