An auto-configuring mesh protocol with proactive source routing for Bluetooth Low Energy

Julio León*, Abel Dueñas, Cibele A. Makluf and Frank C. Cabello

Laboratory of Visual Communications, University of Campinas, Campinas/SP, Brazil
Email: j.leon@decom.fee.unicamp.br
Email: aduenas@decom.fee.unicamp.br
Email: cibeleam@decom.fee.unicamp.br
Email: fcc125@decom.fee.unicamp.br
*Corresponding author

Guillermo Kemper

GIDEPSI, University of San Martin de Porres, Lima, Peru
Email: gkemperv@usmp.pe

Yuzo Iano

Laboratory of Visual Communications, University of Campinas, Campinas/SP, Brazil
Email: yuzo@decom.fee.unicamp.br

Abstract: The Internet of Things (IoT) is spreading rapidly towards creating smart environments. Wireless sensor networks (WSN) are among the most popular applications discussed in IoT literature, with most of them considering many forms of wireless mesh communications. One of the most available and popular wireless technologies for short-range operations (yet not designed for mesh) is Bluetooth. Literature shows some studies on mesh networks BLE, based on Bluetooth 4.1 (which supports Master/Slave Multirole). Those approaches require more powerful hardware than a simple wireless sensor peripheral. Nonetheless, none address dynamic address allocation and topology mapping for BLE. We propose a new autoconfiguring dynamic address allocation scheme for a BLE Ad-Hoc network, and a network map discovery mechanism that does not require role changing, compatible with BLE 4.0 or later versions.

Keywords: Bluetooth Low Energy; mesh; wireless sensor networks; WSN; proactive source routing; auto-configuring; Internet of Things; IoT.

Biographical notes: Julio León obtained his PhD in Electrical Engineering at the University of Campinas (UNICAMP), Brazil, in 2016. He also holds a BSc (2007) and MSc (2011) in Electrical Engineering by the Peruvian University of Applied Sciences (UPC), Peru, and UNICAMP, respectively. His research interests are related to wireless mesh networks, radar signal processing, and the Internet-of-Things.

Abel Dueñas is a PhD candidate at University of Campinas. He holds a Master’s in Electrical Engineering (2016) at UNICAMP. His research interests are related to object recognition, digital image processing, computer vision, and Internet-of-Things.

Cibele A. Makluf graduated in Electrical Engineering at Pontifical Catholic University of Minas Gerais – PUC (2006) and obtained her Master’s in Electrical Engineering at UNICAMP (2011). She is currently a PhD candidate by the Department of Communications. Her research involves 4G LTE networks, digital television, and Internet-of-Things.

Frank C. Cabello holds a BSc in Mechatronic Engineering by National University of Peru – UNI (2009), Lima-Peru, and also holds an MSc (2013) and PhD (2016) in Electrical Engineering by UNICAMP. His research interests are image processing, machine learning, medical imaging, and digital signal processing.

Guillermo Kemper received his BSc degree in Electrical Engineering in 1994 by Antenor Orrego Private University (UPAO), Trujillo, Peru. He holds a Master’s (1994) and a PhD (2001) in Electrical and Communications Engineering, respectively, from UNICAMP. He is currently a Research Professor for the undergraduate and postgraduate programs of the schools of Electrical Engineering from the San Martin de Porres University (USMP), Lima, Peru, and the Peruvian University of Applied Sciences (UPC), Lima, Peru. In addition, he is a Researcher at the National Institute for Research and Training in Telecommunications (INICTEL-UNI), Lima, Peru. His research interests include voice, audio, images and video processing, digital communications, and digital television.

Yuzo Iano is the Head and Founder of the Laboratory of Visual Communications since 1972. He obtained his BSc (1972), MSc (1974) and PhD (1986) in Electrical Engineering at UNICAMP. He is a Full Professor at the Department of Communications, UNICAMP. His research interests are related to smart cities, digital image processing, computer vision, and the Internet-of-Things.

This paper is a revised and expanded version of a paper entitled ‘A Bluetooth Low Energy mesh network auto-configuring proactive source routing protocol’ presented at ICCE 2017, Las Vegas, 8–11 January 2017.
1 Introduction

The Internet of Things (IoT) is spreading rapidly towards creating smart environments. Home automation, intra-vehicular interaction, and wireless sensor networks (WSN) are among the most popular applications discussed in IoT literature. One of the most available and popular wireless technologies for short-range operations is Bluetooth. In late 2010, the Bluetooth special interest group (SIG) launched the Bluetooth 4.0 Specification, which brings Bluetooth Low Energy (BLE) as part of the specification. BLE characterises as being a very low power wireless technology, capable of working on a coin-cell or even by energy scavenging.

On December 2013, the Bluetooth SIG announced the 4.1 specification, enabling connection-based mesh networks (by allowing on-the-fly role switching between master and slave). Some authors have published works considering these new features (Sirur et al., 2015; Reddy et al., 2015; Balogh et al., 2015). However, they all present their results based on more powerful hardware, usually android smartphones. Therefore, an approach compatible with BLE 4.0 (no master/slave role changing) is still viable and less demanding on hardware, which leads to reduced costs for consumer device sensors.

However, devices changing roles between master and slave were not supported until Bluetooth specification 4.1. This limited the options for mesh networking, as any connection-based routing protocol was to be discarded. In the literature, the first attempts on multi-hop communication in BLE (Mikhaylov and Tervonen, 2013) came in the form of massive broadcasting (connection-less). Other authors also presented mesh network topologies based on broadcasting, nRF OpenMesh (Nordic Semiconductor, 2014), and CSR’s CSRMesh (CSR, 2016) (although the latter is closed source).

In 2015, BLEmesh (Kim et al., 2015) was published using opportunistic routing instead of flood routing to reduce unnecessary broadcasts; hence, reducing the overall energy consumption. BLEmesh, based on advertisement broadcasts, is a protocol that manages to transmit information from one node to a destination via broadcasts and opportunistic routing.

The three main problems of any MANET are dynamic address allocation, network topology mapping, and data routing. However, the first two are often assumed in the literature and only the data routing problem is addressed. BLEmesh’s opportunistic routing also requires knowledge of the network map in order to create the priority list, which is assumed by the authors. To the time of this writing, there are no known publications for solving these problems for BLE.

The capability of a wireless technology to create a Mobile Ad-Hoc Network (MANET) is vital for supporting the plethora of sensors, peripherals, and devices that could coexist in any IoT environment. In this work, we propose a new method for dealing with dynamic address allocation and mapping the network topology in BLE. It consists of two parts: an auto-configuration protocol (modified RSVConf) for the nodes to define and maintain IDs; and a proactive source routing (PSR) protocol for discovery, maintenance, and mapping of the network topology in the form of spanning trees. Any ad-hoc routing protocol can utilise the proposed method to discover, keep track, and maintain the mesh network node topology in each of their nodes.
2 Dynamic address allocation

One of the most important features required for a proper IoT wireless mesh network is node discovery and network registration (i.e., the ability for a node to recognise other nodes, and to register itself into the existing mesh network). This is usually done through assigning an address or Node ID to every node forming part of the network using techniques known as Address Allocation. These techniques can be static (i.e., the nodes already know the address they will be assigned *a priori*) or dynamic (i.e., address allocation happens on-demand).

Ad-hoc networks such as MANETs have particular requirements that are not necessarily shared by common centralised networks. These requirements should be addressed by the network protocols, and they consist in (Jeong et al., 2005):

1. setting up the network model
2. allocation of a new joining node with a unique address
3. duplicate address detection (DAD)
4. network address re-utilisation
5. network partitioning and merging.

The network model defines how the ad-hoc network will operate: this is the core of the network protocols. It may define the usage of proxy nodes, broadcast or unicast communications, initial configurations, and network address structures. The model must also consider how to include new elements into the network and make sure they are able to be accessed via a unique identifier inside the network itself; usually by dynamically allocating addresses to them. In some cases, multiple joining nodes may request or be assigned the same address, in which DAD algorithms must be used to ensure that there are no duplicate addresses in the same network. Finally, a network model should suggest at least one form of data routing protocol so that data can be sent from any node in the network to any other node (e.g., ad-hoc on-demand distance vector (AODV) routing) (Jeong et al., 2005).

Almost every method in the literature for dynamic address allocation was designed for IP networks (García Villalba et al., 2011; Nesargi and Prakash, 2002; Sun and Belding-royer, 2003; Weniger, 2005; Zhou et al., 2003), where the maximum transfer unit (MTU) is 7,981 bytes for IEEE 802.11. However, BLE’s advertisement packet maximum payload is only 31 bytes, where 1 byte is reserved by the BLE’s generic access profile (GAP) for the data length and GAP AD (advertised data) fields, respectively. Therefore, the limitations of number of nodes and node ID size will be determined by this.

In 2006, Bredy et al. (2006) proposed RSVconf as a lightweight stateful auto-configuration protocol that designed for highly mobile MANETs; particularly, for the intelligent transportation system (ITS), or inter-vehicular communication based on the internet protocol (Bredy et al., 2006). It supports network partitioning and merging, and has the following features:
An auto-configuring mesh protocol with proactive source routing

- It is a stateful protocol: every node has knowledge of the nodes in the network, as each node maintains and updates a local table with the IP addresses and unique identifiers for every member of the network. This allows any node to manage a new node registration and network mergers and partitions.

- Merger and partition detection: since network IDs (NID) are used on every message, it is easy to detect when a new network collides with a node (receiving periodic broadcasts with a different NID), and process the merger adequately.

- Fast response: depending on the configuration of the protocol (counters and time-outs), it is possible to quickly detect events (e.g., node registration, mergers, etc.) and react to them accordingly.

- Number of nodes: in IEEE 802.11WiFi, the maximum transfer unit (MTU) is 7,981 bytes. With this in mind, RSVconf can support up to 200 nodes without packet fragmentation (considering sending an IP address and a database identifier per node plus a protocol preamble).

- Independence of routing protocol: a routing protocol is not required, as most messages are only sent to the nearest neighbours (1-hop broadcast). Nevertheless, it is possible to take advantage of proactive source routing protocols in the maintenance phase.

In this work, we propose an adaptation of the RSVConf protocol to address autoconfiguration for BLE. We considered appropriate using 7 bits for the node IDs (instead of 32-bit IP addresses). Using packet fragmentation (eight maximum fragments), the RSVConf protocol supports a theoretical maximum of 128 nodes (a reasonable number for an intensive wireless sensor network in a residential environment), and it is still small enough for the Proactive Source Routing protocol to support up to 100 nodes by using packet fragmentation as well (four maximum fragments). Packet fragmentation in RSVConf only occurs in the case of network mergers, as those messages contain the complete ID database.

The protocol consists in three main phases: proxy selection, reservation, and configuration.

2.1 Phase 1: proxy selection

The first phase of the reservation protocol is the proxy selection. In this phase, the joining node (i.e., the new node entering the mesh) auto-assigns itself a 7-bit random ID. Then, it starts broadcasting a Proxy Request (PREQ) message. Other nodes that listen to this (and not currently in a merger or RSV time-out) will respond to the request by broadcasting a proxy offer (POFF).

It is possible that more than one node responds with a POFF message. In that case, the joining node will accept the first POFF and lock onto that proxy, ignoring the other nodes. It will then assume the mesh ID from the proxy (joining the same mesh network) and reply with a Proxy Acknowledgement (PACK) message. Different to the original RSVConf, that required this PACK to be a unicast, BLE reservation requires the PACK to be broadcast as well, but it includes the Proxy’s ID so that other nodes receiving the PACK can ignore it.
If the joining node does not receive a POFF after a certain amount of time, it will assume it is isolated (or could be the first node in a network), and assume a random mesh ID. It will then listen to any PREQ messages. Likewise, if a Proxy does not receive a PACK after a certain amount of time, it will time-out and go back to its normal operation mode. The maximum waiting times are parameters that can be adjusted when fine-tuning for hardware.

2.2 Phase 2: reservation

Once a Proxy receives the PACK, it enters into RSV (reservation) mode. This means it will look up in its local ID database (which is replicated in every node when convergence is reached) for a random available ID. Then, it broadcasts a RSV message with its own ID as srcID, the available ID to be assigned to the joining node as (targetID), and it will also generate a random 8-bit number to identify the node (in case of duplicates), which will be sent in the (Random#) field.

The random number size of 8 bits is sufficient to guarantee a 99.6% chance of two nodes not generating the same number and creating a collision, if they ever where to also randomly assign the same ID to different joining nodes at the same time (e.g. two new joining devices in different rooms).

The RSV message is then rebroadcast throughout the entire mesh network. Each node that receives it checks for the existence of the targetID field. If it exists, it then checks if the targetID already exists in its database. If it does, probably another node was registered with the same ID, so it compares the random numbers. In case of a difference, the Proxy needs to be notified of this, so it can assign a new ID to the joining node. This is done via a Response (REP) message broadcast by the node that detects the conflict.

In both cases, a RSV or a REP, the broadcasting node disables itself from for a short amount of time in order to avoid processing further RSV or REP messages, respectively. This is done so that the broadcasting node will not receive its own broadcast and be forced to rebroadcast it, causing an infinite loop.

2.3 Phase 3: configuration

In the meantime, the joining node processes the RSV message from the proxy, and assumes the assigned ID. It then requests the complete database from the proxy via a REQdb message, so that it becomes fully aware of the other nodes and is able to become a proxy in a very short amount of time since registering. The Proxy will reply with a SENDdb message containing the complete database.

In the case of conflict, the detecting node will respond with a REP message as mentioned above. The REP’s destID field contains the ID of the Proxy that generated the conflict, and the Random# field corresponds to the already existing node (i.e., the correct node), so that the proxy and every node that received the wrong update can correct their database. If the joining node (with the conflicting ID) receives the REP message, it will also compare the Random# field with its own. If it is the same, then it just ignores the REP message; if not, then it looks in the database for a new available ID. If there are no available IDs, it will go back to broadcasting PREQ messages until a new ID is available.

Besides that, the ID database table contained in every node consists of three fields: the IDs, the random number representing the nodes, and a keep-alive counter. At every
timer tick (BLE’s ADV period), the node will increase the counter for every node in the
database. Then, it will check if any of the counters reached a threshold, in which the node
will be considered removed from the mesh network.

In order for the nodes to stay alive in the mesh, they need to periodically broadcast a
keep-alive message, in the form of a RSV message without including the targetID or
the Random# fields. If a node receives a keep-alive RSV message, it will zero the keep-
alive counter from that particular node in its database, and then rebroadcast the message.
Once again, after the broadcast, it will disable itself for a short while to prevent a
broadcasting loop.

2.4 Merger situation

So far, all messages sent and received have been done within the same network
(meshID). If a message is received by a node and the meshID is different than the one
of its own (with the exception of a PREQ message, which does not have any meshID
fields), the node will just ignore that message. However, there is a specific message that
will not be ignored, which is the Detect Merger DM message, which is periodically
broadcast by all nodes with a long period (e.g., 1 second for 20 milliseconds normal
broadcast period).

When a node receives a DM, it checks if the meshID is different from its own. If it is
the same, it then proceeds to calculate an MD5 hash from its database and compare it to
the MD5 hash from the DM message. If the hashes are different, the situation represents a
re-merger (a mesh that was part of the mesh and somehow split). If the meshIDs are not
the same, the process is a two-network merger. The reason MD5 was selected is due its
well known algorithm, and the fact that the hash is barely going to be used for
identification and not for encryption or security.

In any case, the node receiving the DM will compare the size of its own database with
meshSize. If the node’s database size is greater or equal, it proceeds to broadcast a
Merger Hi! (MERHI) message so that the DM sender starts the merger process. If, on the
other hand, the node’s database size is smaller, then it will proceed as if it received the
MERHI itself.

When a MERHI is received by a node, and it is directed to itself, it starts the merger
procedure. First, it changes to the new meshID (in case of a re-merger, this process is
skipped). After that, it adopts the new database as its own, dropping the old one. In case
of a re-merger, it then looks for its own ID and Random#. If they are the same, then it
means that it is the only node with that ID, and it does nothing else. If the Random# is
different, it then looks for a new available ID and adopts it; then broadcasts a Merger
Update (MERUP) message.

The procedure when receiving a MERUP is similar to the MERHI. They are different
messages to avoid multiple mergers at the same time – MERHI is directed, as if it were
unicast, while MERUP is non-directed – .
Finally, when any node executes the merger routine, it disables itself, like in the RSV scenario, so that multiple mergers can be avoided and a convergence can be achieved.

2.5 Packet fragmentation

The maximum number of nodes supported in a single BLE Advertisement packet is defined by (1), where the effective MTU is 29 bytes or 232 bits (elements inside brackets in (1) indicate their dimension or size in bits). This worst case scenario occurs with the largest possible message in the BLE reservation protocol: the MERHI message.

\[
N_{\text{max}} = \frac{\text{MTU}-[\text{msgCode}]-[\text{meshID}]-2\times[\text{nodeID}]}{[\text{nodeID}]+[\text{random#}]}
\]  

Therefore, the maximum number of nodes supported in a single packet for BLE is \(N_{\text{max}} = 13\). In order to support more than 13 nodes, the Database field will require two more fields on the front, as shown in Figure 1. The \(F_n\) and \(F_{\text{max}}\) represent the Fragment Number and Total Fragments per message fields respectively. In a case of approximately 100 nodes, both \(F_n\) and \(F_{\text{max}}\) require 3 bits respectively.

As a special note, it is important to consider that even though this could represent a significant performance loss (augmenting the number of fragments per message), it will only occur on either a merger situation (which would require the propagation of the entire database), or a new node registering (in which only the Proxy would be sending the database). All the other messages do not require sending the database, including the keep-alive messages. Thus, convergence can quickly be reached and maintained with mostly single-fragment messages.

2.6 RSV convergence

The BLE Reservation protocol will continue to transmit and update the node IDs of every node until it has reached convergence (i.e., every possible node in reach is identified and within the same mesh ID). This can be detected by each node through a simple counter mechanism: when a network is converged, only DM and keep-alive messages (RSV) will be circulating in the network. The DM’s will be ignored and not rebroadcast, while the keep-alive will be the only ones being rebroadcast. If a node does not receive any of the other messages after a while, it can decide the network is converged.

3 Proactive source routing protocol

3.1 Background

There are two main types of ad-hoc routing protocols: proactive routing and on-demand (reactive) routing. Proactive routing protocols are table-driven routing protocols that require the mapping of the entire mesh topology. They are called proactive because they periodically update their database with the destinations and routes, and distribute them throughout the network (Shivahare et al., 2012). These types of routing protocols are suitable for small to medium sized networks, as there is the problem of routing overhead
An auto-configuring mesh protocol with proactive source routing

since every node must update the others with their routing table. One of the most popular proactive protocols is destination sequenced distance vector (DSDV).

On-demand routing, on the other hand, are protocols that discover a route whenever they require to send a message. This is done by flooding the nodes with broadcasts requesting route information and relying on each of the node’s routing information to relay the message (Shivahare et al., 2012). Some on-demand protocols include ad-hoc on-demand distance vector (AODV) and dynamic source routing (DSR).

The major problems with the above mentioned protocols regarding MANETs rely on the high susceptibility to node mobility. In order to prevent that, due to the broadcast nature of wireless communications, several nodes can store a packet from a single transmission, and use them as backup. This is known as opportunistic routing, and has been explored in ExOR (Prabhakar and Sikamani, 2013). Moreover, the authors of ExOR conducted a study and developed a very efficient ExOR protocol for vastly dynamic mobile ad-hoc networks based in a new opportunistic routing protocol: position-based opportunistic routing (POR). The basis of this protocol relies on proactive source routing (PSR) (Wang et al., 2014).

The POR protocol presented in (Prabhakar and Sikamani, 2013) is designed for IEEE 802.11 networks, and it would be required to be studied and adapted for BLE. In fact, a similar approach was taken by BLEmesh (Kim et al., 2015), based purely on ExOR and not POR. However, no work has been published considering the underlying requirements for an opportunistic routing protocol in BLE: the actual network topology routing. BLEMesh presents an application-layer protocol for forwarding messages and choosing the best routes, but it assumes the nodes already know all the possible routes and that the nodes have somehow configured themselves. That is the problem this work aims to address.

A lightweight proactive source routing protocol (PSR) was published in 2014 (Wang et al., 2014), demonstrating clearly its advantages over DSDV, DSR, and Optimised LS routing (OLSR). Moreover, distance vector protocols like AODV, which are on-demand, are not suitable for opportunistic routing. Therefore, a modified version of PSR for BLE was the best possible choice for a complete framework for any opportunistic routing protocol.

3.2 Adapted proactive source routing for BLE

The PSR protocol’s goal is to provide each node with knowledge of the entire network topology relative to them. This is done by forming breadth-first spanning trees (BFST) rooted at each of the nodes, with the entire network branching from the root node. In order to build and maintain this tree, the protocol has a route update strategy that bases on events and timers.

Periodically, a node will broadcast its most recent BFST based on the best knowledge it currently has about the network. When a node receives update messages from its neighbours, it updates its own tree structure. Since a node can receive multiple messages before its broadcast timer ticks again, the tree can be updated multiple times in a single BLE Advertisement period. The most updated BFST will be transmitted only at the next broadcast event, not polluting the spectrum with multiple broadcasts.

Finally, in the case of a lost node (disconnected or left the mesh radius), the PSR protocol has the ability to detect and drop the node from its tree, informing the others
during the updates. This, combined with the BLE RSV protocol in database and keep-alive messages, can decide if a node has just moved to another physical location within the network’s reach or it has left the network.

3.3 Tree and route updates

Let the complete mesh network be defined as an undirected graph, \( G = (V, E) \), where \( V \) is the set of vertices (or nodes) in the network, and \( E \) is the set of edges (or links) between each node.

An edge between two nodes (e.g., nodes \( p \) and \( q \)) is defined by the edge \( e = (p, q) \), and \( e \in E \). For any node \( v \), its open neighbourhood (i.e., the set of nodes it can reach in one hop) is defined by (2) every node in its reach. It is important to note that the node \( v \) is not part of its own open neighbourhood (i.e., \( \{v\} \cap N(v) = \phi \)). Likewise, the closed neighbourhood of \( v \), defined by (3), is its open neighbourhood including the node itself.

\[
N(v) = \{u \in V | (u, v) \in E\} \tag{2}
\]

\[
N[v] = \{v\} \cup N(v) \tag{3}
\]

The process of route update starts when the RSV process has converged, either by having all existing nodes already identified and in the same mesh ID, or by being the only isolated node with no neighbours at reach; in this case, \( N[v] = \{v\} \) and \( N(v) = \phi \).

Either way, node \( v \) is only aware of itself in PSR (even though it may have the complete database of every ID in the network), and it becomes the root of its tree. If, in fact, \( N(v) \neq \phi \), the next iteration of broadcasts will provide new information to node \( v \) to update its own spanning tree (a graph rooted at node \( v \)). This spanning tree is denoted as \( S_v \).

Each time node \( v \) receives a message, it theoretically receives the BFST from another node’s perspective. It then has a tree \( T_u \) for each of the neighbours of \( v \), given \( u \in N(v) \). Then, the network topology known by \( v \) so far is constructed by a union graph of \( S_v \) with the trees coming from \( N(v) \), as shown in (4).

\[
G_v = S_v \cup \bigcup_{u \in N(v)} (T_u - v) \tag{4}
\]

The union graph in (4) performs the union of each individual tree coming from each neighbour \( u \in N(v) \), but it removes the sub-tree rooted at \( v \) in \( T_u \). This operation is required as node \( v \) is already its root and it does not require information about its own tree from another node. Furthermore, it is possible to have \( T_u - v = T_w \), which would mean that the arriving tree did not contain information about node \( v \). This usually happens the first time node \( v \) receives a tree update from node \( u \).

3.4 Neighbour dropping

As a form of keep-alive, similar to RSV, node \( v \) keeps track of every node in \( N(v) \), and stores a table with keep-alive counters. Each time it receives an update from any node \( \{u\} \in N(v) \), it resets the counter for that specific node. In the case node \( v \) stops receiving updates from node \( u \) for certain amount of time, the counter will max out, indicating that node \( u \) is no longer a member of \( N(v) \). This could happen due to a number of reasons, for
instance, increased packet loss due to physical obstacles (e.g., a door was closed between nodes), the node was mobile and left the radius of v, the node’s battery ran out, etc. The node update routine consists of removing node u from its own graph Gv. This is explained by Algorithm 1.

**Algorithm 1**  
Node removal routine

Require: Timer event

1. for each row in table containing N(v) do
2. if nodeCount(row) > nodeCountMax then
3.   Remove u from its neighbours: \(N(v) \leftarrow N(v) - \{u\}\)
4.   Construct a new union graph without including information from \(T_u\):
5.   \[G_v = S_v \cup \bigcup_{w \in N(v)} (T_w - v)\] \[(5)\]
6.   Compute the new BFST \(T_v\) from \(G_v\)
7.   else
8.   nodeCount(row) \(\leftarrow\) nodeCount(row) + 1
9. end if
10. end for

Once \(T_v\) is calculated in case of a node time-out, it is not broadcast at that moment. It can still get updated by receiving other trees, and it will only be broadcast during its broadcast event (at BLE’s advertisement period). This keeps synchronisation in the broadcasts and prevents excessive messaging, thus saving power.

### 3.5 Tree representation

The traditional representation of a graph requires a list of its nodes, and the edges between the nodes. If the list of nodes is not present, it can be inferred from the list of edges. Figure 2 illustrates an example of a breadth-first spanning-tree (BFST) graph, rooted at node 1.

However, sending a complete tree with every edge in the graph can result in extremely large update messages and, with a very limited MTU in BLE, can result very inefficient. In Wang et al. (2014), the authors propose a compact tree representation using left-child/right-sibling (LCRS) representation and binary trees. A binary tree is a tree structure in which each element can only have 2 options, or decisions. An example of a binary tree is shown in Figure 3.

**Figure 2**  
Example of a BFST (see online version for colours)
The LCRS representation takes advantage of the binary tree structure and defines the element on the left as a child (tier below), and the element on the right as a sibling (same tier). This means that it is possible to represent a complete BFST with multiple nodes interconnecting in a binary tree form. As an example, the BFST shown in Figure 2 is represented in LCRS in Figure 4(a), or a more compact version for visualisation in Figure 4(b).

Therefore, it is possible to represent an entire tree of the BFST in Figure 4(b) in a serialised form: \([\text{ID}_1]10[\text{ID}_0]01[\text{ID}_2]01[\text{ID}_4]10[\text{ID}_3]01[\text{ID}_5]00\), where \([\text{ID}_1]\) represents the actual binary ID of node 1. In (Wang et al., 2014), the authors use the IP address of each node as their ID, having the complete tree serialised in \(34 \times n\) (32 bits for the IP address plus 2 bits for the child and sibling bits) for \(n\) nodes in the tree.

In BLE’s case, using large IDs is detrimental due to the MTU limitations. For a mesh network supporting approximately 100 nodes, 7 bits per ID will suffice, requiring only \(9 \times n\) bits per serialised tree.
Transmitting the BFST using LCRS represents a data compression gain over using the traditional edges pairs approach of graph theory. In this particular case, where 7 bits per ID are used, a BFST can contain circa 55% more topology information than if it were using the traditional approach.

### 3.6 Message format

The proposed BLE PSR message format is similar to the messages used in RSV, shown in Figure 5. The header starts with a 4-bit message type code, where the code 0000 represents a PSR message. The meshID contains the current node’s network ID, and it is used to avoid any conflicts in case of a new node receiving a PSR message during a merger (very unlikely event since PSR disables when the RSV is not converged). \( F_n \) and \( F_{\text{max}} \) are Fragment Number and Total Fragments, respectively, and are the fields used for Packet Fragmentation. With this 16-bit header, the actual bits available for the serialised BFST are 216 bits, as the maximum effective MTU is 232 bits. Since the tree requires \( 9 \times n \) bits, the maximum nodes per packet would be 24. With 2 bits to determine the fragment number, it is possible for PSR to support a maximum of 96 nodes.

![Figure 5 BLE PSR message format (see online version for colours)](image)

#### Algorithm 2 Serialised BFST composition

```plaintext
1 function Compose BFST(Gv)
2    \triangleright\text{During the Breadth-first search:}
3   out ← \emptyset
4   for each node in Gv do
5      childSibling ← 00
6      if node has child then
7         childSibling ← 10
8      end if
9      if node has sibling then
10         childSibling ← childSibling + 1
11     end if
12     out ← concatenate(out, nodeID, childSibling)
13   end if
14 return out
end function
```

### 3.7 BFST composition

To compute the serialised BFST in LCRS of \( S_v \), the node performs a Breadth-First transversal of its graph, \( G_v \), which includes all the edges known to node \( v \). The
Breadth-First search is performed in ascending order for nodes of the same tier. Performing this search is the equivalent of going through each of the nodes, tier by tier, in a Compact LCRS representation, as shown in Figure 4(b). It is then possible to create the serialised tree by executing Algorithm 2.

Notice that, by definition, the breadth-first search requires the root node (which, in the case of node \(v\), will always be itself (node \(v\)). Due to this fact, the first node ID in a serialised BFST will always be the ID of the source node (the one who sent the message). This is particularly useful to identify the source node for operations such as computing the union graph in (4).

**Algorithm 3** Serialised BFST decomposition

```
function DecomposeBFST(lcrsMessage)

Divide the message into an array of node IDs and childSibling.

```

define nodeArray ← φ

```
childSiblingArray ← φ

```

```
Tree ← φ

```

```
Eeach set of 9 bits in lcrsMessage do

```

```
Append first 7 bits of the set to nodeArray

```

```
Append last 2 bits of the set to childSiblingArray

```

```
end for

```

```
Tree.addNode(nodeArray[1])

```

```
nodeArrLen ← length of nodeArray

```

```
queue ← φ

```

```
currentNode ← nodeArr[1]

```

```
(currentNode keeps track of the node processing the edges.)

```

```
for i ← 2, nodeArrLen do

```

```
▷ If a node has a child, add it to the queue

```

```
if childSiblingArray[i] > = 10 then

```

```
Either 10 or 11 represents a child

```

```
Append nodeArray[i] to queue

```

```
end if

```

```
Tree.addEdge(currentNode, nodeArray)

```

```
▷ If the node has no sibling, change current node and unqueue

```

```
if childSiblingArray[i] = 00 or 10 then

```

```
qLen ← length of queue

```

```
if queue 6 ≠ φ then

```

```
currentNode ← queue[1]

```

```
queue ← queue[2 to qLen]

```

```
end if

```

```
end if

```

```
end for

```

```
return Tree

```

```
end function
```
3.8 BFST decomposition

When a complete message is parsed by a receiving node, it must first decompose the serialised BFST into Graph form $G = (V, E)$ by transforming the LCRS representation into a set of nodes and edges. It is a reverse breadth-first transversal, using the concept of queue as well, as defined in Algorithm (3).

Once the message is decomposed into a BFST (e.g., $T_a$), the union graph can be constructed using (4).

3.9 Packet fragmentation

In the case of a mesh consisting of more than 24 nodes, more than a single fragment is required to completely represent the network; thus, the $F_n$ and $F_{max}$ fields will no longer be always 00. Table 1 shows the possible options using 2-bit fields for $F_{max}$.

Algorithm 4  Fragment processing routine

```
Require: Packet Reception Event

procedure Process Fragments(msgFragment)

msgLen ← length of msgFragment

$F_n ← msgFragment[13 to 14]$

$F_{max} ← msgFragment[15 to 16]$

bfstFragment ← msgFragment[17 to msgLen]

$srcNode ← bfstFragment[1 to 7]$

for each row in tableFragments do

  Look for srcNode in local Fragments Table

  if $srcNode ∈ tableFragments$ then

    if $F_n = tableFragments[srcNode].F_n + 1$ then

      Append bfstFragment to tableFragments[srcNode].message

      tableFragments[srcNode].Fn ← tableFragments[srcNode].F_n + 1

    else if $F_n = 00$ then

      Discard stored fragments for srcNode

      Insert bfstFragments as a new entry in tableFragments.

    end if

  else if $F_n = 00$ then  ▶ If entry didn’t exist

    Insert bfstFragments as a new entry in tableFragments.

  end if

end for

for each row in tableFragments do

  Look for complete messages

  if $tableFragments[row].F_n = tableFragments[row].F_{max}$ then

    TreeUpdate(tableFragments[row].nodeID, tableFragments[row].message)

  end if

end for

end procedure
```
Table 1  Number of fragments per message

<table>
<thead>
<tr>
<th>$F_{\text{max}}$ value</th>
<th>Total number of fragments</th>
</tr>
</thead>
<tbody>
<tr>
<td>00</td>
<td>1</td>
</tr>
<tr>
<td>01</td>
<td>2</td>
</tr>
<tr>
<td>10</td>
<td>3</td>
</tr>
<tr>
<td>11</td>
<td>4</td>
</tr>
</tbody>
</table>

In order to keep track of the messages from nodes in $N(v)$, node $v$ must maintain a fragments table containing the node ID, current fragment, total fragments and the partial message. Once all the fragments are received, it proceeds to construct the message and process it.

Algorithm 4 describes how the message fragments are processed. In the case of single fragment messages, they will always be processed as new. Fragments must be received in order (i.e., 00 → 01 → 10 → 11) to ensure that the reconstructed fragments correspond to the same message. For instance, reconstructing fragments that arrived in the order {01 → 10 → 11 → 00} may produce an incorrect message, as the originating node could have updated its BFST in between messages. Finally, any fragment identified by $F_n = 00$ represents the beginning of a new message. Therefore, if a node receives that fragment, it can discard the old information on the table from that node.

The second FOR loop in Algorithm 4 performs a lookup routine to check for complete messages. If $F_n = F_{\text{max}}$, the message for that particular row (node) is ready to be processed by the Tree Update routine.

3.10 BFST composition with packet fragmentation

When the node composes a new BFST using Algorithm 2, it has knowledge of how many nodes are currently present in the mesh through its tree. If the message requires fragmentation (due to having more nodes than 24), the fragments that follow the first will also contain that node’s ID and a 10 as a header before the serialised BFST fragments in the other fragments. This is done so that the first node ID in any fragment can easily be identified by any receiving nodes and matched with their correspondent entries in the Fragments Table. This means that any message will always have the form of [0000-meshID-$F_n$-$F_{\text{max}}$-nodeID-10-BFST_Fragment].

Tree update processing routine:

```
Require: Call from fragment processing routine

1 procedure Tree Update(srcID, message)
  ▷ For notation purposes, node $u$ is identified by srcID.

2 $T_u \leftarrow \text{decomposeBFST}(message)$

3 $G_v \leftarrow (S_v - \{u\}) \cup (T_u - \{v\})$

4 $S_v \leftarrow \text{composeBFST}(G_v)$

end procedure
```
3.11 Tree update processing

When a complete message is processed, the Tree update routine described in Algorithm 5 is called to update the current graph and compose the newly updated BFST to be sent in the next broadcast. The algorithm is straightforward, as it makes use of the BFST decomposition function described in Algorithm 3.

4 Results

With the design of both BLE’s address allocation and PSR protocols, a proof-of-concept simulation was necessary to validate their functionality. First, each of the protocols had to work independently, and finally an integration of both was in order. Besides that, the simulation required each of the nodes to be working independently from each other.

In order to implement the protocols in a simulation environment, an Object-Oriented Programming approach was taken in MATLAB R2015b. MATLAB, since version R2015b (MathWorks, 2015), introduced Graph and Network algorithm support as part of the main suite. It proved very useful in the understanding and design of the PSR protocol as well as the top-level simulation, thanks to the Object-Oriented Programming support in MATLAB.

The simulation aimed to demonstrate the functionality of the proposed protocols in an application abstraction layer. Thus, complete functionality of the BLE stack below application was assumed, and the broadcast timers were set to match the advertising events depicted in Figure 6. The ADV packet type required is ADV_NONCON_IND, as the protocol does not rely on SCAN_REQ or SCAN_RSP.

Figure 6 Advertising events (see online version for colours)

To test the functionality of the protocols, four separate simulations were performed. The first one included a very basic mesh network consisting of seven nodes and no loops between them, as shown in Figure 7(a).

In the test conditions, each node starts up with a random node ID and a random mesh ID, forcing the RSVconf protocol to work on each of the nodes without having a pre-existing network. Figure 7(b) illustrates each of the nodes’ assigned ID and mesh ID, in the pair (\(X, mY\)), where \(X\) is the node ID, and \(Y\) the mesh ID.
The simulation starts every node in a random order, and the nodes only process BLE RSV protocol messages until they decide the network has converged. After that, the BLE PSR protocol messaging begins.

The second simulation consisted of a simple mesh, but this time included a loop between three nodes [Figure 8(a)]. The purpose of this simulation was to show that loops did not impact the protocol functionality.

The third and fourth simulations considered more applied scenarios: separate rooms with low power devices and one node that linked them to the rest of the network. The third simulation relied on daisy chaining every node in closed physical loops (e.g., rooms in a house), while the fourth one was very similar to the piconets in classic Bluetooth scatternet formation (Zhang and Riley, 2005), with one slightly high-powered node serving as a physical gateway.

The results for all simulations showed complete convergence using the dynamic address allocation protocol. Since the nodes were randomly powered on, there were several cases of mergers and the simulation still converged without issues in several scenarios, including large networks with packet fragmentation. In the PSR simulation, convergence (every node with the correct complete BFST) was achieved for all the same scenarios. For illustration purposes, results for Mesh #2, from Figure 8(a) are illustrated in Table 2 and Figures 9 and 10.

Notice that some of the nodes have not changed node IDs, but have registered in the same mesh network as every other node. One of the reasons for this is that, due to the random events of powering up the nodes, more than one start their own 1-node network before finding a new node requesting a proxy. In some cases, that may never happen, and the first message one of the nodes receives from another is a detect merger (DM) message. When a merger is initiated, the node will make an attempt to keep its ID, if there is no conflict.
An auto-configuring mesh protocol with proactive source routing

Figure 8  Simulation mesh networks, (a) simple mesh with loop (b) real mesh test (c) piconet mesh test (see online version for colours)

It is important to note that these simulations consider the worst-case scenario in both RSV and PSR situations: Every node starts roughly at the same time (minor time differences are simulated in starting times) so that at the beginning of the simulation there are no preformed networks. In these worst cases, the maximum number of network mergers will occur within the RSV protocol.

After the address allocation converges, the same worst-case is simulated for PSR. Every node starts (with proper ID) at roughly the same time, and every node forms their BFST from the ground up. Even in these worst-case scenarios, network convergence was relatively fast, taking but a few iterations to be completely configured.
Although this proof-of-concept was not designed to be a timing simulation, these iteration numbers are directly related to the configurable parameter of the BLE Advertising period. Besides that, even though some of the test scenarios required tinkering in the MATLAB simulations in order to finish the calculations, the proof-of-concept results from all the simulation cases were convincing, showing that both the BLE Dynamic Address Allocation and BLE PSR protocols are suitable for different types of network topologies and can operate with scalability.
5 Conclusions

The two major problems in BLE mesh networking that have been assumed/avoided by the literature so far have been addressed. We have proposed a functional method to support creation of mesh networks for BLE that considers dynamic address allocation and mapping of the network topology, which can be the basis of the development of a complete opportunistic routing scheme for BLE.

Nonetheless, the implementation of the proof-of-concept had some limitations. The choice of the MATLAB environment was useful from an academic point-of-view, allowing to understand and illustrate the complete processes of both protocols. However, it requires heavy computational resources; specially due to the Object-Oriented approach, and simulation of more than 30 nodes becomes very difficult. However, even with those difficulties, results are convincing in showing that the proposed protocols are suitable for implementing a BLE mesh network.

The benefits of this work rely on providing a framework for future routing protocols to use and address new challenges in order to further improve the IoT, and present significant impact on the current IoT Consumer Electronics industry.

References


