Multihop Data Transfer Service for Bluetooth Low Energy

Konstantin Mikhaylov and Jouni Tervonen
RFMedia Laboratory
Oulu Southern Institute, University of Oulu
Ylivieska, Finland
Emails: {konstantin.mikhaylov,jouni.tervonen}@oulu.fi

***Abstract*—Bluetooth Low Energy (BLE) is one of the recently developed protocols enabling energy-efficient short-range radio communication. One of limitations of BLE is the support for data transferring over only a single hop. In the paper, we suggest the mechanism enabling the multihop data transfers between the nodes in BLE networks. To ensure portability, the suggested mechanism is designed as BLE service, which uses only the integral components of BLE stack. We discuss in details the suggested multihop service, its implementation and present the results of evaluation, which confirm feasibility and reveal features of the suggested solution. To the best of our knowledge, this paper reports the first implementation of the multihop data transfer over the BLE networks.**

Keywords—Bluetooth Low Energy; BLE; multihop; network; service; implement; evaluate; test;

I. Introduction

Bluetooth Low Energy (BLE) is the energy-efficient short-range wireless communication protocol, which was introduced in 2010 as a part of Bluetooth Core Specification version 4.0 [1]. The major purpose of BLE developing was to enable products with lower current consumption, lower complexity, and lower cost than the ones possible with the classic Bluetooth technology [2], [3]. During development of BLE and shortly after its introduction, it was predicted that the protocol will find wide application area and will dominate the Wireless Sensor Network (WSN) application market by 2015 [4].

Nonetheless, even today, i.e. three years after the finalization of the BLE specification and two years since the appearance of the first commercial BLE transceivers, only a few real-life applications use BLE. One of the major reasons for this is the imperfections in the current BLE hardware (HW) and software (SW) implementations (see, e.g., [2]). The other reason is the features and restrictions of the BLE protocol. Among the most serious BLE limitations compared to the other protocols is the restriction of the supported network topologies to the single-hop scenarios, i.e. point-to-point or star topologies [2], [5].

To the best of our knowledge, the only previous attempt to solve this issue and to enable multihop (MH) BLE networks was done in [6]. In [6] Haatanen proposed the mechanism enabling the MH BLE network for the specific scenario, when data are transferred in one direction from multiple nodes to the single gateway. Nonetheless, Haatanen was unable to implement and verify the suggested mechanism in practice due to the limitations of the BLE transceivers at the time.

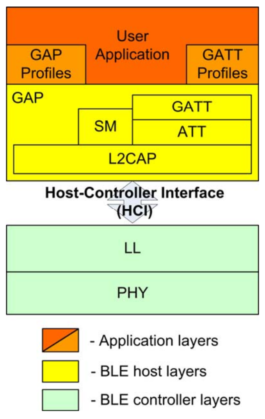


Fig. 1. BLE stack

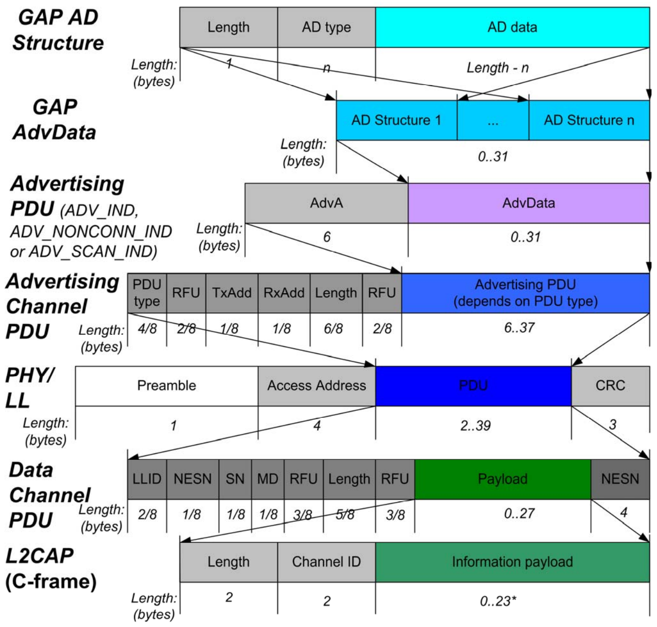


Fig. 2. BLE frame formats (*-23 bytes is the minimum value to be supported as defined in [6], if a BLE transceiver supports higher payloads – it should implement also the apropriate segmentation mechanisms)

Therefore, in the paper, we introduce the novel mechanism, which enables MH BLE networks where data are transferred between any nodes, discuss its practical implementation and present the results of its evaluation.

II. Overview of Bluetooth Low Energy Protocol

As revealed in Fig.1, like the classic Bluetooth, BLE protocol stack consists of the two major components: a BLE Controller and a BLE Host [1]–[3], [5]. The Controller is the logical entity that is responsible for the physical layer (PHY) and the link layer (LL). The BLE Controllers inherit some features from the Controllers of the classic Bluetooth, but both types of Controllers are not compatible [2], [5]. The Host implements functionalities of the upper layers, which are discussed below. Communication between a Host and a Controller is standardized as the Host Controller Interface (HCI) [5]. On a real BLE device the Host and the Controller can either reside on the same physical device or on the different devices.

Similar to the classic Bluetooth, BLE operates in the license-free industrial, scientific and medical (ISM) band at 2.4 GHz. Nonetheless, in order to reduce price and energy consumption, BLE uses binary frequency modulation with 1 Mbit/s over-the-air data rate [1], [2]. Unlike the classical Bluetooth, which utilizes 79 1-MHz-wide channels, BLE uses 40 2-MHz wide channels. The three channels located between the commonly used Wireless LAN channels are employed for advertising and service discovery and are called advertising channels [2], [3]. The remaining 37 data channels are used to transfer data. The frequency hopping mechanism is used on the data channels in order to handle the interferences and improve the communication reliability. Formats of the BLE frames used in advertising and data channels are depicted in Fig. 2. Fig. 2 reveals that BLE uses rather short packets with very limited payload.

The data transfer between BLE devices is bound to the time units called Advertising and Connection events (see Fig. 3) [2]. The Advertising events are used to broadcast the data on the advertising channels. At the beginning of each Advertising event the advertiser sends an advertising frame. To show whether the advertiser can accept the connection establishment requests (i.e. CONNECT_REQ) or the requests for more data (i.e. SCAN_REQ), the advertiser sets one of the four possible Advertising event types. Out of those, three (i.e., ADV_IND, ADV_SCAN_IND and ADV_NONCONN_IND) are capable of transferring the data in the advertisement.

In the case if the advertiser sends the ADV_IND or ADV_DIRECT_IND frames, a device (referred as initiator), which desires to exchange the data with the advertiser can send the CONNECT_REQ frame. If the advertiser accepts the request, the devices continue communication using the data channels (see Fig. 4). In this case, the initiator becomes the master device and the advertiser – the slave. At the beginning of each Connection event (referred to as the Connection event anchor point) the used channel is changed following the predefined sequence [2]. The communication in each Connection event is initiated by the master device. The master and the slave devices alternate sending the frames on the same data channel while any of the devices has data to transmit or until the current Connection event ends [2]. In the case if either master or slave misses a frame or receives two consecutive frames with CRC errors, the current Connection event is closed. According to the specification, minimum period between two consecutive frames on a data channel should exceed the interframe space period (IFS), which equals to 150 µs [1], [2].

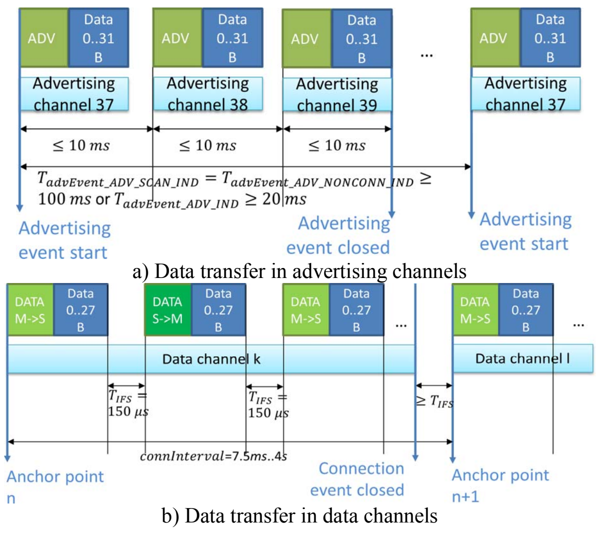


Fig. 3. Data transfer using BLE (number of bytes is given for the LL)

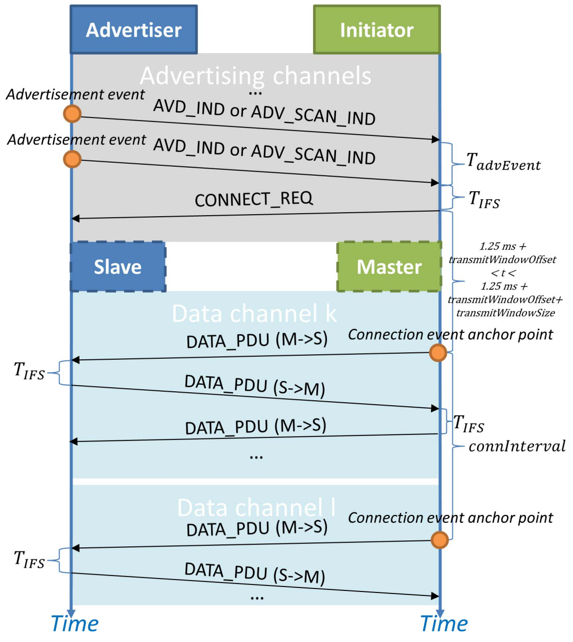


Fig. 4. BLE connection establishing procedure

The Host implements functionality of the upper layers, which include L2CAP, GAP, ATT, GATT and SM [2], [3]. The logical link control and adaptation protocol (L2CAP) multiplexes packets of the upper layers, manages connection establishment, configuration and destruction and can support packet segmentation [1], [2], [5]. In the current version of the specification [1], L2CAP of BLE supports only three types of traffic, which are distinguished using the L2CAP channel identifiers (CID) in the L2CAP packets (see Fig. 2). The supported types of traffic are: the signalling traffic (CID=0x0005), the traffic for attribute protocol (ATT) (CID=0x0004) and the traffic for security manager (SM) (CID=0x0006) [1], [3]. The SM manages authentication and pairing of devices, bounding, and encryption [2]. The attribute protocol ATT defines client-server mechanisms, which enable discovering, reading, and writing of the attributes. The generic attribute profile (GATT) is based on ATT and provides the framework for discovering the various characteristics (i.e. the data units) and the services (i.e. sets of characteristics and references to other services) [2], [5]. The actual user BLE application usually operates on top of GAP and GATT layers (see Fig. 1). Note that although the general structure of the stacks for BLE and classical Bluetooth is similar, the functionality of the layers differs dramatically. Also the specification prescribes each Bluetooth device to have a unique 48-bit IEEE 802-2001 address for distinguishing and addressing the device [1].

As one can see, the BLE only enables the communication between the devices located within direct communication range and does not support any MH data transferring [2], [5], [7].

III. Suggested Mechanism for Multihop Data Transfering in BLE Networks

To enable transferring the data over multiple hops in a BLE network two mechanisms are required. The former one should handle discovery and establishment of connections with the neighboring nodes and handle discovery of a route to the distant nodes. The latter one should handle storing of the transferred data on the intermediate nodes and handle the transmission of the data further towards the destination node.

There are two major ways, how the MH data transferring for BLE can be implemented. The first option is to introduce the MH traffic as a novel BLE L2CAP traffic type using one of the currently reserved CIDs (similarly to what has been suggested for transferring the IPv6 traffic over BLE in [8]). Although this would enable one to develop a completely new protocol, which can be optimized for MH data transferring, the protocol cannot be used in the real-life applications until the standardization and official CID assignment by Bluetooth SIG. Therefore, we chose the other option and implemented the MH transferring as a special "Multihop Transfer" service (MHTS) on top of the GATT layer.

A. General Architechture

The MHTS has four characteristics (see Table 1). The first one is the write-only characteristic *route_entry*, which is used by nodes for transferring the data about the routes. The second characteristic is the read only *own_addr*, which stores the 6-byte address of current device. The third characteristic is the 6-byte final destination address *dst_addr*, which can be both read and written. The last characteristic is the write-only *data*, which is used for transferring the data and should be capable of storing the maximum payload one can put into a

TABLE I. GATT Characteristics of MH transfer Service

Characteristic	Permissions	Size, bytes	Description
route_entry	Write	14	Used to transfer data about the routes. Includes address of the target node, address of the next-hop node and the total number of hops to the target node.
own_addr	Read	6	Address of this node.
dst_addr	Write&Read	6	Address of the final target device for the block of data.
data	Write	>20	Part of data to be sent to *dst_addr*.

TABLE II. Format of MH Service Internal Structures

Structure name	Structure entries[a]	Size[b], bytes	Description
SeekTbl	*SeekAdr*	6	Address of the node to which the route should be found
	NumHops	2	Maximum number of hops
	SeekTime	2	Remaining time to seek route to *SeekAdr*
RouteTbl	*FinalDstAdr*	6	Address of the final destination node
	NHAddr	6	Address of the next-hop node towards *FinalDstAdr*
	HumHops	2	Number of hops up to *FinalDstAdr*
DataBuf	*FinalDstAdr*	6	Address of the final destination node
	Data	var[c]	Data to be transmitted to *FinalDstAdr*

a. The number of entries in each structure depends on implementation.
b. For a single entry.
c. Depends on the implementation.

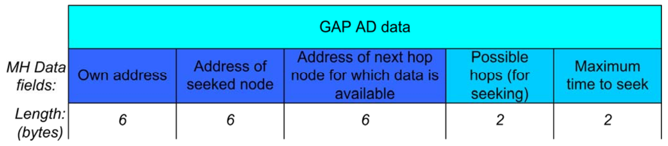


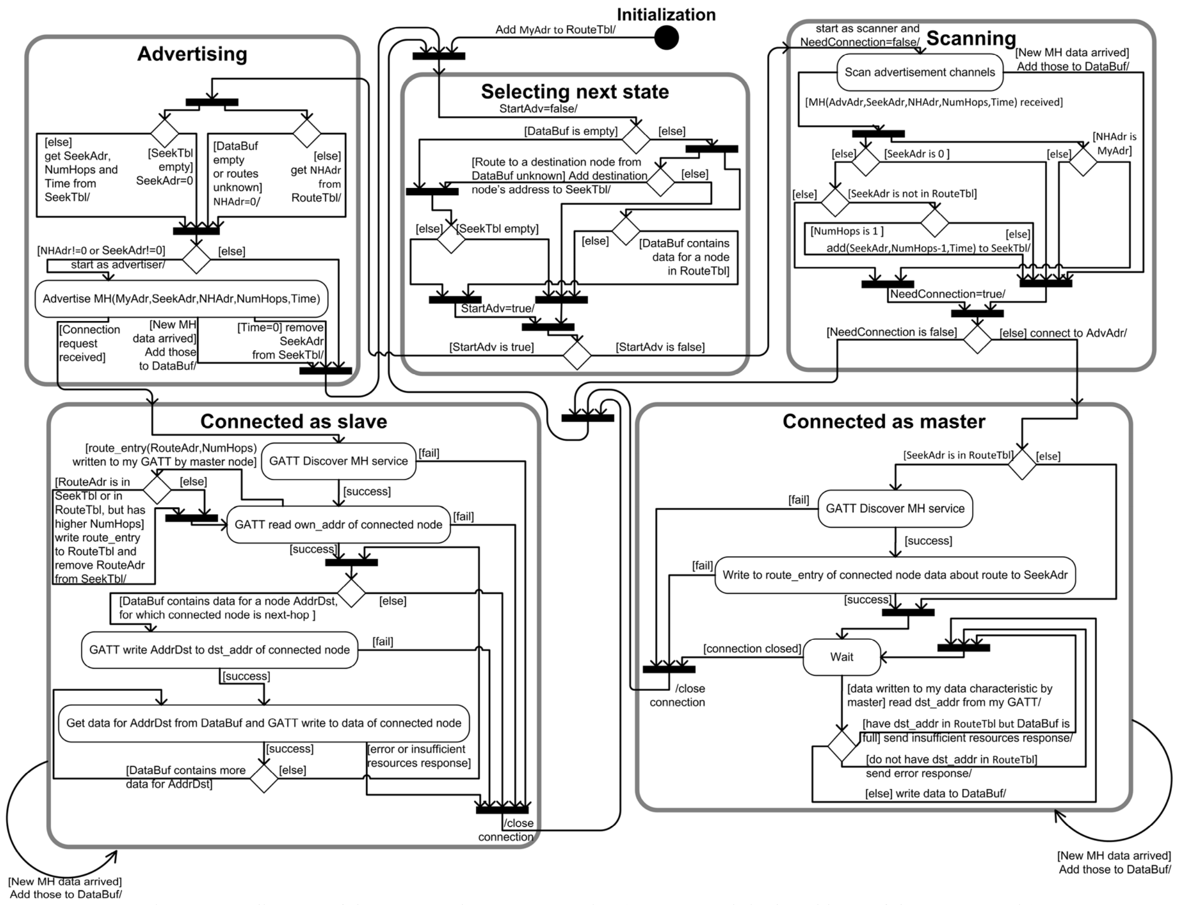
Fig. 5. Format of GAP advertising data (see Fig.2) for MHTS

GATT write (GATT_WR) command (i.e., at least 22 bytes according to [1]). Also we consider that there is an internal mechanism, which informs the application layer implementing the MHTS that GATT characteristics were modified by a distant node.

Besides four GATT characteristics, the MHTS has three internal structures, which are depicted in Table 2. The first structure is named *SeekTbl* and stores the addresses of the nodes, the routes to which are currently being discovered, and the parameters of the search. The second structure is called *RouteTbl* and stores the data about routes to all known nodes. The table stores the address of the final destination, the address of the next-hop node and the remaining number of hops. Note, that the first entry in *RouteTbl* is added during the service initialization and contains the address of the node itself. The last structure is the *DataBuf*. The *DataBuf* stores as well the data to be transmitted using the MHTS and the address of the final destination node.

The format of the GAP advertising data (AD) used by MHTS is depicted in Fig.5. The brief state diagram of MHTS is presented in Fig.6. The operation of the MHTS is illustrated in Fig.7 and discussed in Sections III.B and III.C.

Note that we assume that the note initiating the MH data transferring always knows the address of the destination node.



Initialization
Add MyAdr to RouteTbl/
Advertising
Selecting next state
StartAdv=false/
Scanning
start as scanner and NeedConnection=false/
[New MH data arrived] Add those to DataBuf/
Scan advertisement channels
[MH(AdvAdr,SeekAdr,NHAdr,NumHops,Time) received]
[else] get SeekAdr, NumHops and Time from SeekTbl/
[SeekTbl empty] SeekAdr=0
[DataBuf empty or routes unknown] NHAdr=0/
[else] get NHAdr from RouteTbl/
[NHAdr!=0 or SeekAdr!=0] start as advertiser/
Advertise MH(MyAdr,SeekAdr,NHAdr,NumHops,Time)
[Connection request received]
[New MH data arrived] Add those to DataBuf/
[Time=0] remove SeekAdr from SeekTbl/
[DataBuf is empty]
[Route to a destination node from DataBuf unknown] Add destination node's address to SeekTbl/
[SeekTbl empty]
[DataBuf contains data for a node in RouteTbl]
StartAdv=true/
[StartAdv is true]
[StartAdv is false]
[SeekAdr is 0]
[NHAdr is MyAdr]
[SeekAdr is not in RouteTbl]
[NumHops is 1]
add(SeekAdr,NumHops-1,Time) to SeekTbl/
NeedConnection=true/
[NeedConnection is false]
[else] connect to AdvAdr/
Connected as slave
[route_entry(RouteAdr,NumHops) written to my GATT by master node]
GATT Discover MH service
[fail]
[success]
[RouteAdr is in SeekTbl or in RouteTbl, but has higher NumHops] write route_entry to RouteTbl and remove RouteAdr from SeekTbl/
GATT read own_addr of connected node
[DataBuf contains data for a node AddrDst, for which connected node is next-hop]
GATT write AddrDst to dst_addr of connected node
Get data for AddrDst from DataBuf and GATT write to data of connected node
[DataBuf contains more data for AddrDst]
[error or insufficient resources response]
/close connection
[New MH data arrived] Add those to DataBuf/
Connected as master
[SeekAdr is in RouteTbl]
GATT Discover MH service
Write to route_entry of connected node data about route to SeekAdr
[connection closed]
Wait
[data written to my data characteristic by master] read dst_addr from my GATT/
[have dst_addr in RouteTbl but DataBuf is full] send insufficient resources response/
[do not have dst_addr in RouteTbl] send error response/
[else] write data to DataBuf/
/close connection
[New MH data arrived] Add those to DataBuf/

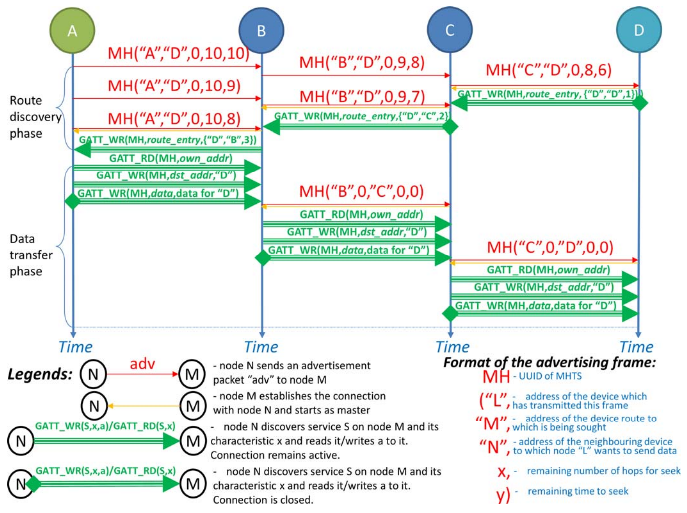
Fig. 6. State diagram of the suggested BLE MH service (Note: MyAdr is the address of the current node)

B. Nodes and Routes Discovery

Once MHTS has data for a distant node and does not know the route, the discovery procedure is started. The MHTS adds the address of the target node in the *SeekTbl* and defines the parameters of the search (i.e. maximum number of hops and the time for seeking) based on the priority of the data. After this, the node starts sending the advertising frames, which include the data from the *SeekTbl* (see Figs. 5-7). The neighboring BLE nodes with active MHTS, which currently listens at the advertising channels, receive the advertising packet. In the case if a node does not have the data about the sought node in its *RouteTbl* – it adds the data from the received advertising packet into its *SeekTbl* (if both remaining time and number of hops are non-zero). If a node knows the route to the sought node, it establishes the connection to the node from which the advertisement was received and starts as connection master. Once the connection is established, the master discovers the MHTS on the slave and writes in the slave's *route_entry* characteristic the data about the route to the sought node (see Table 2). The slave node inserts the data received in the *route_entry* in its *RouteTbl* and excludes the address of the found node from its *SeekTbl*.

C. Multihop Data Transfering

Once the MHTS has data to send to a distant node and knows the route, the data transfer procedure is initiated. First of all, the service obtains the address of the next-hop node from the *RouteTbl*. The address is then included in the special field of MHTS's advertising data (see Figs. 5 and 7). Once receiving the advertising frame, the node with this address establishes connection with the advertiser. The slave node (i.e., former advertiser) discovers the MHTS on the master and reads the *own_addr* from the master to ensure that the connection has



A
B
C
D
MH("A","D",0,10,10)
MH("A","D",0,10,9)
MH("A","D",0,10,8)
MH("B","D",0,9,8)
MH("B","D",0,9,7)
MH("C","D",0,8,6)
GATT_WR(MH,route_entry,("D","D",1))
GATT_WR(MH,route_entry,("D","C",2))
GATT_WR(MH,route_entry,("D","B",3))
GATT_RD(MH,own_addr)
GATT_WR(MH,dst_addr,"D")
GATT_WR(MH,data,data for "D")
MH("B",0,"C",0,0)
MH("C",0,"D",0,0)
Route discovery phase
Data transfer phase
Time
Legends:
adv
- node N sends an advertisement packet "adv" to node M
- node M establishes the connection with node N and starts as master
GATT_WR(S,x,a)/GATT_RD(S,x)
- node N discovers service S on node M and its characteristic x and reads it/writes a to it. Connection remains active.
- node N discovers service S on node M and its characteristic x and reads it/writes a to it. Connection is closed.
Format of the advertising frame:
MH - UUID of MHTS
("L", - address of the device which has transmitted this frame
"M", - address of the device route to which is being sought
"N", - address of the neighbouring device to which node "L" wants to send data
x, - remaining number of hops for seek
y) - remaining time to seek

Fig. 7. Illustration of MH Service operation. Node A discovers the route and sends data to node D through nodes B and C.

TABLE III. MAIN CHARACTERISTICS OF CC2540 SOCS & RADIO MODULES USED IN TESTS[9]

Device type	System-on-Chip (BLE transceiver + 8051 microcontroller)
Microcontroller specification	Clock: 32 MHz Flash: 256 kB RAM: 8 kB
Radio protocol and stack version	BLE (TI-BLE v1.3.1)
Operating radio band	2.4 GHz
Modulation	GFSK
Spectrum spreading	FHSS
Over-the-air data rate, kbit/s	1000
Transmit power range, dBm	-23..4
Receive sensitivity, dBm	-87 (standard mode)
Supply voltage, V	2-3.6
Sleep current consumption [a], µA	0.9

a. With active timer.

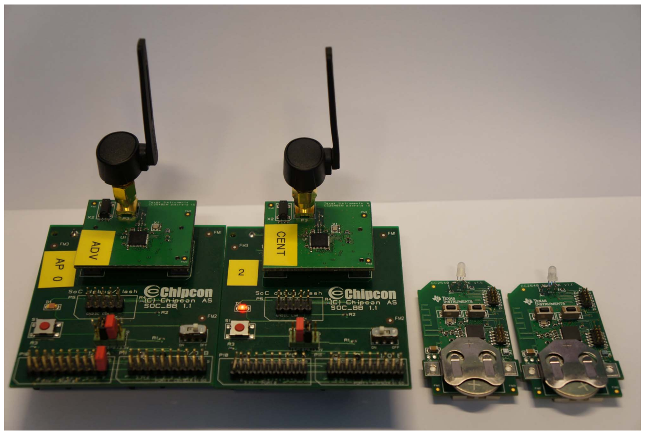


Fig. 8. Hardware modules used for the tests (left to right): two CC2540EMK boards (used as the first and the last node) and two CC2540DK-mini boards (used as intermediate nodes)

been established correctly. Then the slave writes the address of the final destination node for the transferred data in the *dst_addr* characteristic of the master. After successfully writing the *dst_addr*, the slave starts getting the data for the *dst_addr* from its *DataBuf* and writes those in the *data* characteristic of the master. The data are written in the blocks not exceeding the supported by the master maximum GATT characteristic size (see Section III.A). In the case if the master does not have sufficient resources to store the received data in its *DataBuf*, the node sends the "insufficient resource" response. Otherwise, if the data was successfully written to *DataBuf*, the node acknowledges the reception of the data. The slave continues sending the data to the master while it has in the *DataBuf* the data to be transferred to the nodes, for which the master node is the next-hop node, and while the master node has sufficient resources to store the data.

Note, that immediately after the reception of a new *route_entry* (see Section III.B), the slave node checks whether it has the data that might be forwarded via the master node (see operation of node A in Fig.7).

As one can see in Fig. 5, the advertising data of the MHTS can contain simultaneously two different addresses – the one of a node the route to which must be discovered and the address of a neighbouring node, through which the advertising node wants to send the data.

IV. IMPLEMENTATION AND EVALUATION OF THE MULTIHOP DATA TRANSFERING SERVICE

For evaluating the feasibility and the features of the suggested BLE MHTS, we have implemented and tested it in a real-life experiment. The service was developed on top of the Texas Instruments' (TI) BLE stack (version 1.3.1), which was running on the TI CC2540 systems-on-chips (SoCs)[9]. The main characteristics of the CC2540 SoCs are summarized in Table 3.

Table 3 reveals that the CC2540 SoCs combine both the 8051-based microcontroller core and the BLE-compatible radio transceiver. The TI BLE stack provides the solution, which enables the microcontroller core to control the BLE transceiver and implements the functionalities of both the BLE Host and the Controller (refer to Fig.1). The suggested MHTS was implemented on top of the GATT and GAP layers provided by the stack.

To test the MHTS we used four hardware modules based on the CC2540 SoCs (see Fig. 8). The nodes were placed at a distance of around one meter from each other. The frame source filtering was used to force the nodes to transfer the data over the three hops (see Fig.7: e.g., node B is accepting the packets only from nodes A and C, but not from node D). For broadcasting the MHTS's advertisements (see Fig.5) we used the ADV_IND events with the minimum allowed advertising interval (i.e. 20 ms [1], [2]). Also, for transferring data on the data channels we have used the minimum possible value for the *connInterval* (i.e. 7.5 ms [1], [2]).

In the beginning of the test we have placed on the first node (node A on Fig.7) the block of data destined for the last node (node D on Fig.7). As the routing tables of all nodes were cleaned prior to the start of the test, the nodes firstly had to discover the route and then to transfer the data over the three hops. The successful reception of the data was signalized by the light-emitting diodes on the last node.

V. DISCUSSION AND LESSONS LEARNED

The most significant challenges during the implementation of the BLE MHTS were:

- the realization of switching between the roles of advertiser and scanner for a node;
- the deficiency of the RAM memory available on the nodes.

The TI BLE stack was developed in the first place keeping in mind the applications where a node acts all the time either as the advertiser/connection slave or as the scanner/connection master. Meanwhile, as discussed in Section III and revealed in Fig. 6, the MHTS requires a node to periodically switch between the roles of advertiser and scanner. To implement the role switching we had to rethink the node's initialization procedures and the architecture of the application of the TI BLE stack.

One of the consequences of supporting both the advertiser/connection slave and the scanner/connection master roles for the same node is the high consumption of the resources. The total size of program code of the BLE stack (with maximum code optimization by the compiler) was around 136 kB, which is around 52% of the total available code memory (see Table 3). Also, while supporting both roles, the BLE stack and the underlying OSAL operation system (OS) reserved around 7.6 kB of the random access memory (RAM), which is 95% of the total RAM. Even though the 2772 B of RAM is reserved for OS's memory heap and around 1100 B can be further reused by the MHTS, one can see that the total amount of data memory available for the service is rather scarce. Therefore, for our tests we limited the maximum number of the entries in the *SeekTbl* and *RouteTbl* to five and the maximum size of data buffer to one kilobyte. Nonetheless, we do not consider this issue to be critical because the available for a node RAM can be easily increased by adding an external memory chip.

Although our experiments confirmed the feasibility of MH data transferring in BLE networks in general and specifically the suitability of the suggested MH service for this purpose, those have revealed one significant limitation for MH data transferring in BLE. Even though in our experiments we used the minimum possible values for advertising and connection intervals, the total time for finding the way and sending one data block of 13 bytes over three hops in practice ranged from 20 to 30 seconds (25 s in average). Note that the time for sending more than one packet does not increase significantly. E.g., the average time for finding the way and sending 40 13-byte packets was 28 s. Meanwhile, once the route is known, the time for sending the data over the three hops is between 6 and 9 seconds. There are two major reasons for this. First of all, once two nodes establish the connection, those have to use the appropriate ATT mechanisms firstly to discover the MH service and then to find the associated with the service characteristics. This requires sending back and forth multiple frames. Meanwhile, as discussed e.g., in [2], [5], the TI BLE stack can send only a few frames in one connection event. Therefore, e.g., the transmission of the *route_entry* over each hop requires as many as 12 connection events (i.e. 90 ms for minimum possible *connInterval* value) after the connection establishment. The transmission of data over each hop requires 21+2**n* connection events (i.e. 157.5+15**n* ms), where *n* is the number of the packets (up to 22-byte payload). As easy to see, if not to consider the time spent for advertising, the discovery of a route and transmission of a single data block of up to 22 bytes over the three hops will require at least 787.5 ms.

Nonetheless, in our experiment the major time the nodes were spending in the advertising mode. It could require up to ten seconds from the start of advertising up to the establishment of connection between an initiator and an advertiser. We suppose that the major reason for this is the specifics of the implementation of these procedures in TI BLE stack (some features of the stack are discussed e.g., in [2], [5]).

VI. CONCLUSIONS

Bluetooth Low Energy (BLE) is the novel energy-efficient short-range wireless communication protocol. The previous studies (e.g., [2]) have shown that compared with the other technologies, BLE provides an inexpensive and power-efficient solution for wireless communication.

In the paper, we have suggested, discussed the practical implementation, and presented some evaluation results for the special multihop (MH) service, which enables MH data transfer in the BLE networks. To the best of our knowledge, the suggested MH service is the first solution enabling MH data transfer between any nodes in BLE networks. Also, to the extent of our knowledge, the suggested solution is the first one, which has been implemented and tested in practice.

The suggested MH service is designed as a service, which uses only integral components of the BLE stack. The service can be implemented on any BLE device, which can act as a scanner and as an advertiser, and can establish the connections in the data channels. To prove the feasibility of the MH data transfer using the suggested solution, we have implemented and tested the service using the CC2540 SoCs from Texas Instruments. In our experiments, we successfully used the developed MH service implementation to transmit the data over two and three hops. Nonetheless, the presented in the paper results show that the MH data transfer in the BLE networks requires quite significant time.

Although we have proved that MH data transfer in BLE networks is possible and have introduced a solution for this, there are still many open problems. First of all, in our test we could use only four nodes, which is definitely not enough to reveal all possible networking effects. To reveal those, the extensive study including the simulation of the multi-node BLE networks is required. One of the major challenges here is the absence of simulators capable to simulate BLE at the time. Another very important issue is how it is possible to provide sufficient security during MH BLE communication and how to handle mobility of nodes. Finally, as energy efficiency is one of the major targets for BLE, it is necessary to study how efficient the suggested approach is and how the sleeping nodes in the BLE networks will affect the MH data transfer.

Nonetheless, we suppose that even in the current form the suggested MH data transfer mechanism for the BLE networks can be used in practice and can enable the development of novel and exciting applications.

REFERENCES

[1] *Bluetooth Specification*. Core Package version 4.0, 2010.

[2] K. Mikhaylov, N. Plevritakis, and J. Tervonen, "Performance analysis and comparison of Bluetooth Low Energy with IEEE 802.15.4 and SimpliciTI," *J. Sens. Actuator Networks*, vol. 2, no. 3, pp. 589–613, Aug. 2013.

[3] R. Heydon, *Bluetooth Low Energy: the developer's handbook*. Upper Saddle River, NJ: Prentice Hall PTR, 2012.

[4] "WTRS Wireless Sensor Network Technology Trends Report," West Technologies Research Solutions, Mountain View, CA, USA, WT062510CNTS, Jun. 2010.

[5] C. Gomez, J. Oller, and J. Paradells, "Overview and Evaluation of Bluetooth Low Energy: An Emerging Low-power Wireless Technology," *Sensors*, vol. 12, no. 9, pp. 11734–11753, Aug. 2012.

[6] M. Haatanen, "Self-organizing routing protocol for Bluetooth Low Energy sensor networks," B.Sc. thesis, Oulu Univ. Appl. Sc., Oulu, Finland, 2012.

[7] E. Mackensen, M. Lai, and T. M. Wendt, "Performance analysis of an Bluetooth Low Energy sensor system," in *Proc. 1st Int. Symp. Wireless Syst. (IDAACS-SWS'12)*, Offenburg, Germany, 2012, pp. 62–66.

[8] H. Wang, M. Xi, J. Liu, and C. Chen, "Transmitting IPv6 packets over Bluetooth low energy based on BlueZ," in *Proc. 15th Int. Conf. on Adv. Commun. Technol. (ICACT'13)*, Pyeongchang, ROK, 2013, pp. 72–77.

[9] "2.4-GHz Bluetooth Low Energy system-on-chip," Texas Instruments, 2013.