A Server Application for Bridging the TCP/IP and Bluetooth Protocols

Lubomir V. Bogdanov[1], Ratcho M. Ivanov[1] and Nikolay B. Iliev[2]
1) Department of Electronics, Faculty of Electronic Engineering and Technologies
Technical University of Sofia
8 Kliment Ohridski blvd., 1000 Sofia, Bulgaria
{lbogdanov, r.ivanov}@tu-sofia.bg
2) Nextlab Ltd, 8 Kliment Ohridski blvd., 1000 Sofia, Bulgaria
nik@nlab.bg

Abstract – **The following paper discusses the implementation of a software application meant to fill the gap between the Bluetooth and the TCP/IP protocol. This would allow to expand the wide area network Internet further by introducing new types of devices that currently exist in small local networks – battery-powered embedded devices. The application aims to make a protocol translation.**

Keywords – **Bluetooth; embedded systems; Internet-of-Things; server application; TCP/IP.**

I. Introduction

The trend of connecting mobile and household devices has been growing ever since the microcontroller industry started integrating Ethernet and Wi-Fi modules. As widely known, the Internet-of-Things term can be used to describe this connection boom. However, a certain class of low-power embedded devices remains in the periphery – the Bluetooth devices.

The structure of the Bluetooth is to use point-to-point or point-to-multiple-points connection [1]. Currently, a mesh of Bluetooth Low Energy (BLE) devices is starting to emerge [2] as a new technology to interconnect multiple low-power devices in the field. The Bluetooth specification is constantly evolving and always targets battery-operated devices that cannot afford the energy budget of an IEEE802.11 (Wi-Fi) link. Other communication protocols also exist, such as 6LoWPAN, Thread, Zigbee and Matter, however they are not as popular and not as widely used as the Bluetooth.

Thus, a disconnection between the two types of networks, low-power and high-power, is created. The proposed software in this paper aims to diminish this gap and bring the Bluetooth devices to the global network Internet. The method is depicted in Fig. 1.

As a high-power host a personal computer, a server computer, or a single-board computer could be used. The requirements for it is to have an active internet connection, a Bluetooth adapter and one of the following operating systems – Windows or Linux. The host must be supplied from the power grid – currently it has the greatest energy requirements of all the nodes in the network.

The nodes depicted as HP are the "high-power nodes" that could or could not be powered from the grid. Usually these nodes represent household items such as smart TVs, tablets, laptops, refrigerators, washing machines, etc. The link between them and the host device is a fast link – usually in the range 100 – 1000 Mbit/s. This is the reason for their high-power demand.

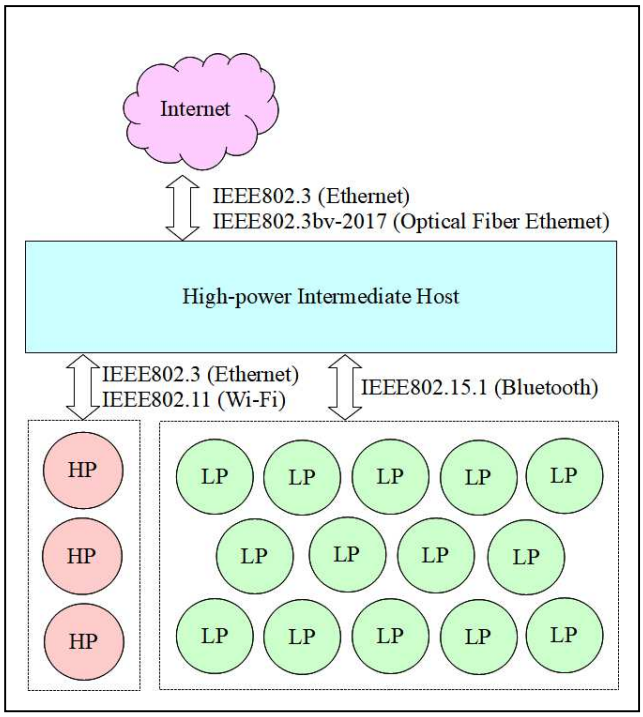

Fig. 1. Connecting Bluetooth devices to the internet through a high-power host. Abbreviations are LP – low power node, HP – high power node.

The nodes depicted as LP are the "low-power nodes" and are battery operated. Such nodes could be smart garden automation devices, home automation devices, etc. The link between them and the host is typically up to 2 Mbit/s. The microcontrollers that support Bluetooth are designed to be low power with idle consumption of approximately 10 µW and 10 mW during transmission [3]. Devices using such integrated circuits (IC) are powered by a single or dual alkaline 1.5V batteries, or a 2032 Li-Ion battery. The former have a capacity of typically 1 Ah, the latter – 0.225 Ah. The time period between changing batteries is long – 1 or 2 years at least.

Once the connection is established, the LP nodes and the HP nodes can communicate freely between each other, and between them and other nodes that are available on the internet. This transforms the local Bluetooth network to a wide area network. The final goal is to get the advantage of the low-power Bluetooth communication and the world-

979-8-3503-0200-4/23/$31.00 ©2023 IEEE

wide accessibility of the TCP/IP communication. This should expose the Bluetooth device control to any point in the world, making it enter the IoT domain.

II. Related work

The idea of connecting Bluetooth devices to the internet is not new. In [4] such an implementation is thoroughly discussed. Authors noted that the error rates in the wireless transmission could be overcome by using the TCP Vegas congestion control scheme. A simulation model is created between two nodes – a laptop and a server. The IP layer is connected to the Bluetooth's L2CAP layer and a buffer holds the data to be transmitted. Once the IP layer writes some data, the L2CAP sends it and removes the entry from the buffer. Up to 700 kbit/s throughput has been measured.

The author in [5] has described the incompatibility of the BLE and TCP/IP. He discusses an implementation with a Bluetooth Internet gateway to cope with the problem. The author uses existing software technologies like the Apache Web Server for the TCP/IP communication, and the BlueZ libraries/configuration tools for the Bluetooth communication. The gateway is actually not a hardware, but a software script written in Python. The author uses a Raspberry Pi demo board that has the needed hardware to make the bridging.

The presented work in [6] shows how to integrate Bluetooth connectivity into an existing TCP/IP implementation of a classical Ethernet switch. Authors showed how the problem with oversized implementations, such as PCs with Bluetooth dongles, could be solved by substituting them with a single network switch.

Though the mentioned research is relevant and very well carried out, it has some drawbacks. The work shown in [4] relies on modification of the data link layer (L2CAP) which means that whenever one of the two protocols changes, this modification has to be corrected also. The work shown in [5] and [6] relies on many third-party libraries and utilities that change between the releases of the Linux distributions. This means that the "software adapters" have to be updated accordingly.

The presented work in the current paper lacks these drawbacks and differs in two ways – it is independent of the changes in both protocols (as long as the operating system has them integrated), and it is compiled as a single executable program – no additional libraries and tools are needed. An additional advantage could be noted – the program is cross-platform and can be run on Linux and Windows. An option to support Mac OS also exists.

III. Implementation details of the server application

The block diagram of the server is shown in Fig. 2. The Qt Creator development environment and framework has been used for the creation of this tool. It is written in C++. Though the binaries are 64-bit, they could be rebuilt for 32-bit machines easily.

The core of the program is the command line parser. It receives user commands over the TCP connection and executes them. To start the server application one may type in a terminal the following string:

```
./blue_tcpip -port 5555 -ip_addr 192.168.0.100
```

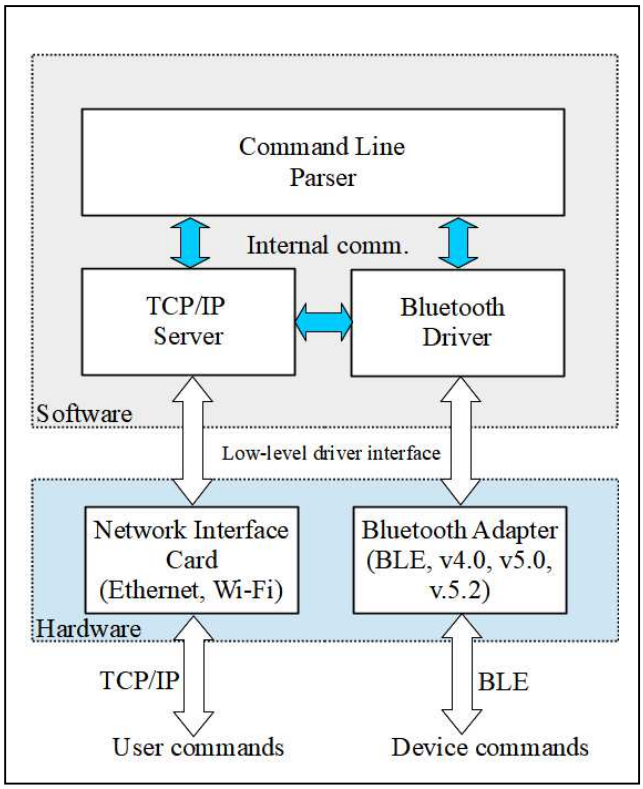

Fig. 2. Block diagram of the server application.

Here the "-port" argument defines the port at which the server should be bound, and "-ip_addr" argument is the IP address of the host machine. If the "ip_addr" argument is not specified, the server tries to guess the IP. If it cannot find a valid address, the server exits with an error. If the given port is already occupied by another server, an error is displayed.

After the successful initialization of the server, the Bluetooth driver is instantiated on the heap, and the hardware BLE adapter is initialized. The Qt Framework is responsible for the low-level initialization and communication, which gives the cross-platform status of the project. No hardware-dependent code could be found in the main application.

Once initialized, the TCP/IP server is the link between the user and the Bluetooth devices. All commands are executed by the user, and all replies are with respect to the user commands. The only exception from this rule is the "disconnected" event that is asynchronous to the command flow. It depends on the hardware of the connected BLE device – if, for example, the power is removed, the disconnection event will be sent to the user notifying him/her that no further commands could be executed.

To make the tool universal, the commands and data are sent in text mode, i.e. no raw bytes could be seen. However, raw bytes could be sent to and could be read from the BLE peripheral by specifying the bytes as text where two ASCII chars form the digits in hexadecimal representation (digits and the letters A, B, C, D, E, F), and they are followed by the space character to separate the bytes. Supporting text and this pseudo-raw representation would make any high-level application that uses this server easy to implement.

An example where those two modes could be used, would be a device that plays sound. One of the communication channels could be used for commands like turn volume up,

play stereo or mono, mute, etc. The other channel could be used for the sound samples. The commands could be sent as text, while the sound samples could be sent as raw data.

IV. The Bluetooth Driver

The Qt development environment supports a cross-platform C++ classes that can be used to communicate with a BLE device on any of the Qt's supported platforms. The classes are the following:

- QLowEnergyController
- QLowEnergyService
- QLowEnergyCharacteristic
- QBluetoothDeviceInfo

but to include a Bluetooth support in the Qt project, the following line has to be added to the ".pro" file, as well:

QT += bluetooth

The current state of this library of classes has one minor disadvantage - the main actor in the communication, an object of type QLowEnergyController, can only make a connection to a single peripheral device. This is due to the fact that the events, called signals in Qt, can be sent only in response to a single peripheral. There is no method of the class that can distinguish between the signals sent from different peripherals.

To support multiple access (in a star topology), an array of controller classes with a higher abstraction has to be created. The index of the array would be the index of the connection that is associated with a single peripheral, as shown in Fig. 3. This would allow the server to be used in multi-threaded environments, which is a major requirement in the modern software.

Each controller object has an array of classes for the services, characteristics and the descriptors for a single connection [7]. Each controller emits private signals within the higher-level class and this is how the peripheral activity is separated. To support the single TCP/IP channel for communication with the user (as shown in Fig. 2), each message is prepended with three numbers – device index, service index and characteristic index. Next, the message is transmitted. This works both for the peripheral and the central data flow.

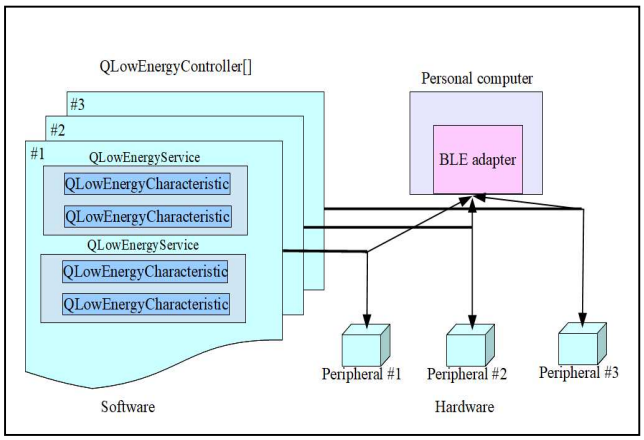

Fig. 3. Raising the level of abstraction of the QLowEnergyController class.

V. Supported Commands

To effectively support all features of the BLE communication, message passing to all of the logical endpoints has to be implemented for the service and the characteristic. Even though the descriptor can be implemented as a characteristic (from software point of view), separate commands for it are created. A list of SCPI-compatible commands is given in Table 1.

Table 1. Supported Commands

Command	Argument	Reply	Description
STSC	[num]	[output] + DONE	STart SCanning for BLE devices for 'num' of milliseconds.
CONN	[num]	DONE	CONNect to BLE device 'num' from the device list.
SRVD	[num]	[output] + DONE	Start SeRVice Discovery for device 'num'.
CHRD	[num0] [num1]	[output] + DONE	Start CHaRacteristic Discovery for device 'num0' and for service 'num1' from the service list.
DCED	[num]	DDFC	Disconnect CEntral from Device 'num'.
DWRI	[num0] [num1] [str]	DONE	Descriptor WRIte data. Here 'num0' is the device index, 'num1' is the service, 'num2' is the descriptor index from the characteristic list, 'str' is the data to be written.
CWRI	[num0] [num1] [num2] [str]	DONE	Characteristic WRIte data. Here 'num0' is the device index, 'num1' is the service, 'num2' is the characteristic, 'str' is the data to be written.
EXIT	-	-	Terminate the Bluetooth-TCP/IP server.
LIST	-	[output] + DONE	Show a list of BLE connected devices.
RAWD	[0/1]	DONE	Set data format, 0 - use strings, 1 - use binary.
TEST	-	TEST	For testing purposes, reply with the string 'TEST'.

HELP	-	DONE	Print all of the commands to the user.
*IDN?	-	Nextlab Ltd,Bluetooth-to-TCP/IP bridge,1.3	Get the Bluetooth-TCP/IP server software revision number.

Some of the commands could fail. In those cases, the string "ERRO" is returned. Peripheral device replies are prepended with the string "REPL", so that the controlling application could parse their output easily.

The scanning for the BLE devices command returns a list with nearby devices and their radio emission power in dBm. The closer the device to the central is, the more positive (towards 0) the power reading is:

```
STSC 5000
0|-68| MAC address shown here |Bluetooth
1|-73| MAC address shown here |Water Vasko
2|-77| MAC address shown here |Water Prof
3|-73| MAC address shown here |Bluetooth
DONE
```

In the excerpt above, the closest device is device #0, with power reading of -68 dBm. The fields of the STSC command are separated with the '|' character and start with device index, given by the server program. This index is used in every other command as a first argument, e.g. if a connection to device "Water Prof" is needed, the command "CONN 2" should be executed, and so on. The second field is the measured power. The third field is the MAC address of each peripheral (hidden in the excerpt for security reasons). The fourth, and final, field is the string that is being advertised by the peripheral.

To be able to write to a characteristic, a certain set of commands has to be initiated and their ordering must be preserved:

- peripherals discovery;
- connection to one or multiple peripherals;
- service discovery;
- characteristic discovery.

If each phase goes without errors, the command for writing has the following format:

```
CWRI 3 2 2 Peripheral command with arguments
```

that in this example means - device index 3, service index 2, characteristic index 2.

To allow for peripheral replies reception, a descriptor has to be written with the appropriate values (usually 0x0001), prior to characteristic writes. The replies have the following format:

```
REPL 3 2 2 Peripheral reply as a string
```

that in this example means – a reply from device number 3, service 2, characteristic 2 has been received.

VI. Conclusion

The paper introduces a new method for bridging TCP/IP and Bluetooth traffic. A server program that supports both protocols channels messages from the internet to Bluetooth devices. The server program is implemented in a low-level abstraction and must be used with a higher level application to ease the user.

Future improvement may include several new features. A high-level GUI program should be developed to provide interface between the user and the server. The peripheral indices for service and characteristics should be replaced with device UUID numbers [8] that would make easier the integration of the server with higher level applications (they will not be forced to track indices internally, but just an UUID number). Another update would be to use separate TCP/IP ports for each peripheral connection (this would separate the data flow between them). And as a final addition – a command to use one of multiple Bluetooth adapters would be useful because some hosts have more than one such adapters (e.g. integrated one and a USB dongle).

Acknowledgment

The authors would like to thank the Research and Development Sector at the Technical University of Sofia for the financial support.

References

[1] U.F. Khan, S. Hameed, T. Macintyre, *TCP/IP over Bluetooth*, Sobh, T. (eds) Advances in Computer and Information Sciences and Engineering. Springer, Dordrecht. https://doi.org/10.1007/978-1-4020-8741-7_85, 2008.

[2] J.V. Marcelo Paulon, B. Olivieri de Souza, M. Endler, *Exploring data collection on Bluetooth Mesh networks*, Ad Hoc Networks, Volume 130, 102809, ISSN 1570-8705, https://doi.org/10.1016/j.adhoc.2022.102809, 2022.

[3] T. Caroff, S. Brulais, A. Faucon, A. Boness, A. Arrizabalaga, J. Ellinger, *Ultra low power wireless multi-sensor platform dedicated to machine tool condition monitoring*, Procedia Manufacturing, Volume 51, Pages 296-301, ISSN 2351-9789, https://doi.org/10.1016/j.promfg.2020.10.042, 2020.

[4] N. Johansson, M. Kihl, U. Körner, *TCP/IP over the Bluetooth wireless ad-hoc network*, Lecture Notes in Computer Science 1815:799-810, Proceedings of conference NETWORKING 2000, Broadband Communications, High Performance Networking, and Performance of Communication Networks, IFIP-TC6 / European Commission International Conference, DOI: 10.1007/3-540-45551-5_67, Paris, France, May 14-19, 2000.

[5] Abhimanyu Rathore, *Understanding Bluetooth internet gateways for IoT solutions*, https://iot.electronicsforu.com , online, 2021.

[6] V. Schuermann, T. Mann, A. Buda, J. Wollert, *Integrating Bluetooth localization into existing TCP/IP networks*, 2009 IEEE International Workshop on Intelligent Data Acquisition and Advanced Computing Systems: Technology and Applications, ISBN:978-1-4244-4901-9, DOI: 10.1109/IDAACS.2009.5342903, Italy, 2009.

[7] C. Liu, Y. Zhang, H. Zhou, *A comprehensive study of Bluetooth Low Energy*, Journal of Physics Conference Series 2093(1):012021, DOI: 10.1088/1742-6596/2093/1/012021, 2021.

[8] Yakov Shafranovich, *Bluetooth data exchange between Android phones without pairing*, Networking and Internet Architecture (cs.NI), arXiv:1507.00650, online, 2015.