TinyBlue: A Bluetooth LE Module for TinyOS

Thomas Bauer
Stanford University
Department of Electrical Engineering
Stanford, USA
tbauer01@stanford.edu

Dave Deriso
Stanford University
Institute for Computational and Mathematical Engineering
Stanford, USA
dderiso@stanford.edu

***Abstract*— Recent advances in Bluetooth low energy (BLE) technology have enabled long term wireless connectivity with extremely efficient power management. This technology has sparked a fast growing trend in small connected sensor-based consumer electronics based on BLE. The present work integrates of Bluetooth low energy and TinyOS, a power efficient operating system for distributed sensing.**

Keywords—TinyOS; Bluetooth; Low Energy; BLE

Bluetooth Low Energy (BLE) is an exciting technology that enables reliable wireless communication with minimal power requirements to common devices. BLE has sparked a fast growing trend in small connected sensor-based consumer electronics. Most of these use the same basic architecture: a sensor, a microcontroller, and a BLE transceiver. However, little work has been done to integrate BLE into existing power-efficient harware and software plaforms, such as wireles motes running TinyOS. The present work aims to integrate BLE into TinyOS and contribute to a very power efficient distributed communication platform. We begin with a survey of wireless technologies, an closer look at the BLE protocol, and an overview of our integration with TinyOS. A demo application and source code are publicly available on GitHub (*http://bit.ly/TinyBlue*).

I. A Brief History of Low Energy Radio

A. Origins

Norman Abramson, a professor at the University of Hawaii, developed the world's first wireless computer communication network, ALOHAnet in 1971 [1]. The system was based on low-cost Ham radios, and was deployed on seven computers spread across four islands to communicate with the central computer on the Oahu Island without using phone lines. Since then, wireless networking has become smaller, more efficient, and ubiqutous in modern electornics.

B. Wi-Fi

In 1991, AT&T invented WaveLAN, the precursor to Wi-Fi, which was intended for use in cashier systems. In 1997, the IEEE developed the 802.11-1997 standards for wireless networks and later branded it as "Wi-Fi," since it sounded better than "IEEE 802.11b Direct Sequence." The original specification operated in the 2.4GHz spectrum and provided for 1-2 Mbit/sec, but was later upgraded to 11 Mbit/sec in 1999.[2] The latest approved standard, 802.11af, achieves 568.9 Mbit/s for 8 MHz channels, while the not-yet-approved 802.11ax wifi standard proposed by the Huawei Corporation has demonstrated a whopping 10.53 Gbit/sec on the 5 GHz band! [3]

A Wi-Fi device consumes approximately 116 mA at 1.8 V (116 mA x 1.8 V = 0.210 W) when transmitting a 40 Mbps User Datagram Protocol (UDP) payload, achieving a power per bit of 0.210/40M = 0.00525 μW/bit. Although Wi-Fi is a very efficient wireless technology, it is designed for transferring large amounts of data at a high-speed, and is not designed to be run from a coin cell. Unfortunately, current consumption does not reduce when throughput is reduced. [4]

C. Zigbee

ZigBee was established in 2003 as a low-power wireless specification (IEEE 802.15.4-2003). It was designed for mesh networking deployable sensors, such as smart meters, home automation, and remote control units. [5] A Zigbee device consumes 0.035706 W when transferring 24 bytes of data (192 bits), translating to a power per bit = 0.035706/192 = 185.9 μW/bit. One issue with ZigBee radios is that

they do not hop frequencies and are therefore susceptible to interference.

D. *Near-Field Communication*

Near-field communication (NFC) was established in 2004 with the ISO 13157 standard. It was designed to be an extension of radio-frequency identification (RFID) to enable two-way interactions with high security due to the fact that it only works with very close (near-field) range of 5-10cm. NFC operates at 13.56MHz at rates ranging from 106 kbit/s to 424 kbit/s. [6] NFC consumes approximately 15mA at 1.8 V (15 mA x 1.8 V = 0.027 W), which achieves a power per bit = .027/424 = 0.064 µW/bit.

E. *ANT*

ANT was developed in 2004 by the sensor company Dynastream, and for some reason there is no explanation for what ANT stands for. It was designed to enable sports and fitness sensors to communicate with other devices, such as a watch or cycle computer. ANT+ has standardized the ANT protocol to made devices from different manufacturers interoperable. [7] An ANT device is configured to transmit 256 bits/second and consumes 61 µA (3 V x 61 µA = 0.183 µW), thus achieving a power per bit = 0.183 µW / 256 bits = 0.71 µW/bit. Due to the low power consumption, ANT devices may operate for years on a coin cell.

TABLE I. COMPARISON OF WIRELESS STANDARDS

	Year	*µW/ Bit*	*µW*	*Bit/Sec*	*Cost*	*Frequency*	*Range*
Wi-Fi	1997	0.00525	210	40 M	$9	2.4 GHz	200m
Zigbee	2003	185.9	35.7	250 K	$2	2.4 GHz	100m
NFC	2004	0.064	27	424 K	$3	13.56 MHz	10cm
ANT	2004	0.71	183	256	$3	2.4 GHz	100m
BLE	2006	0.0004	96	480	$3	2.4 GHz	100m

F. *Bluetooth*

Bluetooth low energy started in 2006 at Nokia Research with the goal of enabling small devices to be permanently connected to the Internet while only being powered by a coin cell battery. It was originally called "Wibree," and was renamed Bluetooth "Ultra-Low-Power" and then Bluetooth low energy, though it is sometimes called Bluetooth "Smart." [8] It's governed by the Bluetooth 4.0 specification, while classic Bluetooth is 1.0-3.0. Data packets (30 bytes) are broadcast every 500 ms, consuming 16µA at 3V (nRF8001 chip), and achieving a power of 16µA x 3V/ 500ms = 0.096W, and power per bit rate of 0.096W/(30 x 8) = 0.0032. BLE was designed to run on a coin cell and most development kits come set up this way.

G. *Bluetooth Low Energy vs Classic Bluetooth*

While BLE has many similarities with its predecessor, it departs from classic Bluetooth in many ways. For one, BLE has a different protocol that is not compatible with classic Bluetooth devices, unless the BLE chip is a "dual" mode device. While Bluetooth classic focused on a strict set of use cases like audio and data transfer, BLE was designed for simplicity and extensibility, so that a developer could interface with any accessory without having to know a great deal about the underlying technology. In addition, BLE was designed to have the lowest possible power consumption, optimized for low cost, low bandwidth, low power, and low complexity and designed to easily exchange data. [9] BLE also has an inherent resource requirement assymetry where masters, such as smartphones and tablets, require more resources, while slaves, such as beacons, use far less resources enabling them to run on inexpensive microcontrollers and radios with smaller power supplies.

H. *Protocol Stack*

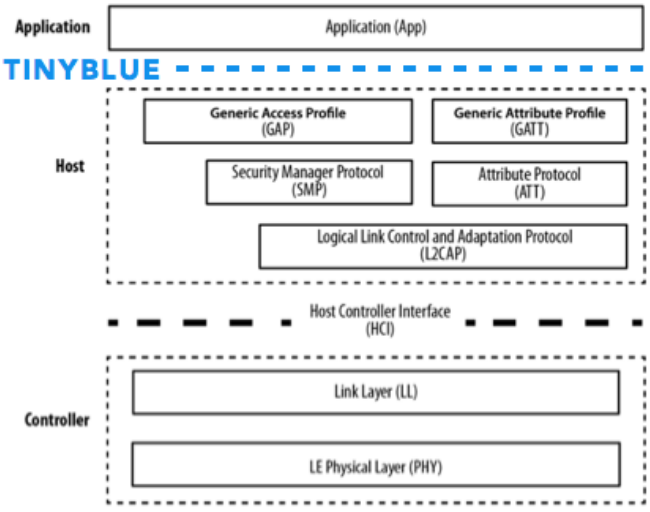


Fig. 1. The Bluetooth LE Protocol Stack (adapted from [10])

The BLE protocol stack consists of three main components: (1) Application manages the actual use case (logic, user interface, and data handling), (2) Host containing the upper layers of the Bluetooth protocol stack, and (3) Controller containing the lower layers of the Bluetooth protocol stack including the physical radio. [10] These can be integrated into the same chip or broken into separate ICs and connected via a standard Host Controller Interface (HCI), which allows interoperability between hosts and controllers produced by different companies. We will proceed to explore the details of how the stack functions from the bottom up.

The physical (PHY) layer contains the physical communications circuitry, which modulates and demodulates analog signals into digital symbols at 1Mbit/sec (which fixes the upper bound on the output rate) using Gaussian Frequency Shift Keying (GFSK, which uses Gaussian filter to smooth positive/ negative frequency deviations, which represent a binary 1 or 0). The digitial signal is over the 2.4 GHz ISM (Industrial, Scientific, and Medical) band, which is divided into 40 channels. To avoid interference from other radios, BLE uses *frequency hopping spread spectrum* where the data channel is shifted by some arbitrary constant whenever it encounters resistance. Since there are 40 channels in the band and 3 are used for the advertiser signal, the shift is calculated as *new channel* = (*current channel* + *hop*) mod 37.

The Link Layer (LL) is a combination of hardware and software that interfaces with PHY and manages the timing, connection parameters, encryption, advertising, scanning, starting, and stopping connections as well as the "link state," which characterizes the role of the device. The roles include: Advertiser: blindly sends advertising packets before active connection without knowledge of the presence of a scanner, Scanner: blindly scans for advertising packets before active connection without knowledge of an advertiser, Master: initiates a connection and manages it later, and Slave: accepts a connection request and follows the master's timing. The LL also checks packets received against a 24-bit CRC, and requested a re-send whenever an error occurs; the LL on the master will resend the packet until it finally passes the test on the receiver.

The Host Controller Interface (HCI) is a standard serial protocol for host-controller communications with a pre-defined set of commands and events, a data packet format, flow control rules, and other procedures that vary by transport, such as UART, USB, SDIO, etc.

The Logical Link Control and Adaptation Protocol (L2CAP) breaks down data from the top layers into chunks that fit in the 30 byte BLE packet size for the link layer. It also recombines chunks of data from incoming packets into a single larger packet to send up the layers to the application.

The Security Manager Protocol (SMP) generates and manages security keys for an encrypted communications link. It stores remote device IDs and hides the public Bluetooth Address to prevent the device from being tracked by unauthorized peers.

The Attribute Protocol (ATT) manages the client and server actions, which do not depend on whether the device is in a master or slave state. The pathway is quite simple: a client requests data from a server and a server sends data to clients, however no additional requests can be sent until a response is received and processed (and vice versa). Processed data is organized into attributes that are indexed by a 16-bit universally unique identifier (UUID), each having a value and a set of permissions (read, write, etc). The Generic Attribute Profile (GATT) is a layer of abstraction on top of the ATT that defines how data in the ATT is organized and transferred between applications.

The Generic Access Profile (GAP) is the topmost layer and ensures interoperability between devices from different vendors. It orchestrates the lower level functions such as discovery, connecting, data transfer, and security.

I. Packets

Data packets have a usable data payload of 27 bytes, but are depending on whether upper layers of the stack take up space. The actual amount can be around 20 bytes per packet. For example: the L2CAP packet header takes up four bytes, which means that the effective user payload length is 27 - 4 = 23. It's worth noting here that the BLE has two kinds of packets, advertising and data, but only one format, which simplifies the implementation.

Packets are sent at a fixed intervals, where shorter intervals increase the probability of those packets being received with the cost of higher power consumption. A scanner also listen for packets at set intervals, which also has an impact power consumption since it varies the amount of time the radio must be turned on.

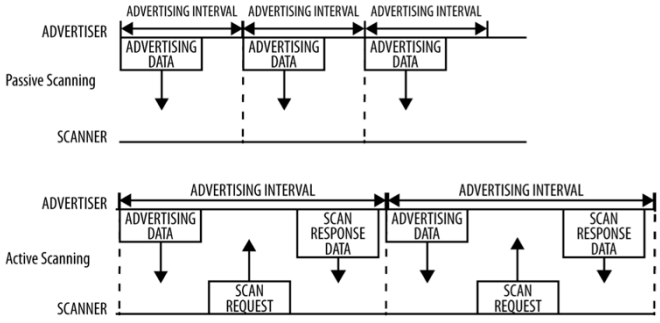


Fig. 2. Handshake Timeline [10]

Advertising packet types can be classified by three properties: connectability, whether a scanner can connect on receipt; scannability, whether a request scan packet can be sent; and directability, whether a packet is sent to a particular scanner. The scanner also has two distinct modes: passive, where the advertiser is unaware if a packet is received, and active, where the scanner requests a "scan response" packet upon receipt.

J. *Finding a Connection*

A Bluetooth LE connection has three components: advertising, scanning, and connecting. The process is much like males seeking females at a bar. The slave device (male) sends advertising data (fancy drinks) that indicate a desire to connect. The master device (female) picks up these packets but filters them by Bluetooth Address or the advertising data itself (slightly degrading for the male). If the master is actively scanning (think Thursday night at the Rosewood), it will let the slave know when a connection is desired. In this case, when a suitable slave is detected, the master (female) sends a connection request packet to the slave (telephone number) and, provided the slave responds, establishes a connection (celestial spiritual bond). The master's connection request packet includes the frequency hop increment (pace of the relationship), which determines the hopping sequence that both the master and the slave will follow during the lifetime of the connection (happily ever after).

K. *Nordic nRF8001 Hardware*

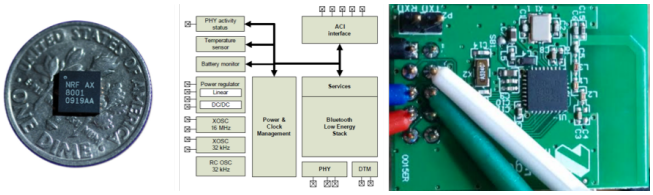


Fig. 3. nRF8001 Chip, Stack, and Dev Board [11]

Nordic Semiconductor was a member of the board that defined the core BLE standard from the very beginning and designed one of the first affordable BLE peripheral-mode chips, the nRF8001. Nordic was kind enough to sponsor this project by sending a nRF8001 development kit, which was used to build our system. The firmware is quite complicated and requires a specialized tool chain (Kiel) that costs several thousand dollars a seat. We avoided using this by using the default firmware and sending a pre-defined boot sequence to the chip, which is explained below.

II. TinyBlue Module

A. *TinyOS*

TinyOS is described as "an operating system designed for low-power wireless embedded systems. Fundamentally, it is a work scheduler and a collection of drivers for microcontrollers and other ICs commonly used in wireless embedded platforms." [12] It is written in nesC, an extension to C that is tailored to the needs of TinyOS.

The project originally sought to add BLE chip and corresponding driver to TinyOS platform and allow applications to access it via the familiar active message interface. Once the BLE platform had been constructed the power consumption was to be measured and compared to the original radio. Unfortunately the use scenarios for BLE and the CC2420 style radio do not compare nicely, largely due to differences in protocol.

The projected shifted to integrating the nRF8001 and Nordic BLE stack into TinyOS to allow motes to communicate over BLE, this could not be a substitute for the original radio as was envisioned. TinyOS was built on radios that broadcast openly or to specific nodes but a connection did not have to be established ahead of time in either case. Whereas

BLE is a connection based protocol with a client and a server model, albeit an interesting one since the connection it initiated by the server. So rather than attempt to connect the BLE stack into the active message interface and compare power consumption the project focused adding BLE functionality to TinyOS as an expanded feature and not a substitute for the original radio.

B. *"Micable" Platform*

A new platform was constructed, micable, which was an adaptation of the micaz platform to incorporate the new nRF8001. At the time the micaz was selected because it was believed that this was the only mote platform that made SPI lines externally available via the MDA100 expansion board developed by Crossbow for the mica platforms. It was later discovered that the only SPI line on the expansion board was SCLK, so the SPI was bit banged using the GeneralIO interface connected at the platform level the necessary IO lines.

C. *Bluetooth Application Control Interface*

The Nordic stack consisted of two main parts, high and low-level platform independent software drivers, and two parts that were used for data encoding and queuing. Each part of the nRF8001 stack provided by Nordic became a tiny OS interface and a top level driver configuration was used to wire them together and ultimately to the lowest platform dependent layer (*see fig.2*).

lib_aci: This was the top level provided by the nRF Bluetooth stack. It was used to determine what commands would be sent to the *hal_aci_tl* interface.

hal_aci_tl: This module was the lowest platform independent level of the driver. It was told what command to send by *lib_aci* and determined how the commands would be sent. It used pins set at the platform level to bit bang SPI commands out that were placed in the TX queue and add incoming message to the RX queue.

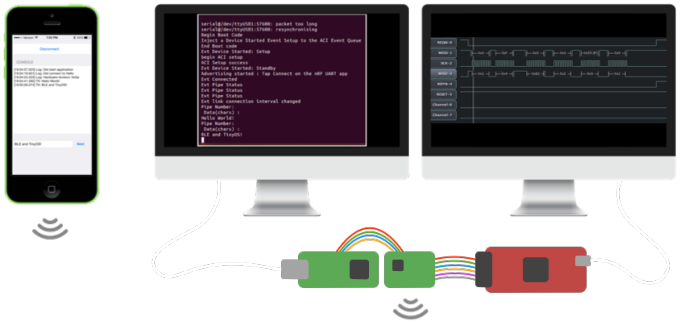
acilib: This module encoded standard messages defined by the application command interface.

aci_queue: This module was used to perform operations of the incoming and outgoing queues to that application command interface.

D. *TinyOS Demo App*

A UART test application was adapted from the Arduino library to run in Tiny OS which allowed the mote to successfully send messages back and forth between an iPhone and the Micaz mote. The mote is the client and the iPhone is the server in this configuration. Once the server has initiated a connection it can send information over BLE that the mote will print to the terminal via a serial port. A logic sniffer (red device in *fig.3*) was attached to the SPI lines to verify that both devices were communicating as expected.

Fig. 4. The Demo Application Setup



The demo application prints out messages to the terminal indicating which step is occurring:

```
java net.tinyos.tools.PrintfClient -comm serial@/dev/tty.usbserial-XBTGCED5B:micaz

      Thread[Thread-2,5,main]serial@/dev/tty.usbserial-XBTGCED5B:57600: resynchronising
      Begin Boot Code
      Inject a Device Started Event Setup to the ACI Event Queue
      End Boot code
      //pump in the firmware
      Evt Device Started: Setup
      begin ACI setup
      //set up ACI
      ACI Setup success
      Evt Device Started: Standby
      Advertising started : Tap Connect on the nRF UART app
      Evt Connected
      Evt Pipe Status
      Evt Pipe Status
      Evt Pipe Status
      Evt link connection interval changed
      Pipe Number:
      Data(chars) : Hello World!
```

For more detailed information, please refer to the actual codebase, which is publicly available on GitHub (*http://bit.ly/TinyBlue*).

III. Conclusion

This project demonstrated that Bluetooth Low Energy can be integrated into the TinyOS system. The combination of low energy devices and software is a useful combination for researchers who demand ultra-low power sensing applications. Future work should be geared towards implementing TinyOS on BLE SoCs so that only

one IC is needed for the OS, sensing, and communication.

The authors wish to thank Professor Philip Levis for his help and support, and Nordic Semiconductor for their generous support.

REFERENCES

[1] Abramson, Norman. "THE ALOHA SYSTEM: another alternative for computer communications." In *Proceedings of the November 17-19, 1970, fall joint computer conference*, pp. 281-285. ACM, 1970.

[2] http://en.wikipedia.org/wiki/IEEE_802.11

[3] http://www.ispreview.co.uk/index.php/2014/05/next-generation-10gbps-wifi-5ghz-sucessfully-tested-due-2018.html

[4] http://www.digikey.com/en/articles/techzone/2011/aug/comparing-low-power-wireless-technologies

[5] http://en.wikipedia.org/wiki/ZigBee

[6] http://en.wikipedia.org/wiki/Near_field_communication

[7] http://en.wikipedia.org/wiki/ANT_(network)

[8] http://en.wikipedia.org/wiki/Bluetooth_low_energy

[9] http://en.wikipedia.org/wiki/Bluetooth

[10] K. Townsend, C. Cufi and R. Davidson, *Getting Started with Bluetooth Low Energy: Tools and Techniques for Low-Power Networking*. 2014.

[11] http://www.nordicsemi.com/eng/nordic/download_resource/17534/14/47977595

[12] http://tinyos.stanford.edu/tinyos-wiki/index.php/TinyOS_Overview