The Benefits of Open Source Bluetooth Stacks

An Emerging Class of Open Source for MCUs: Apache Mynewt and NimBLE

Aditi Hilbert, Head of Product, Runtime
Sterling Hughes, CTO, Runtime

September 11, 2017

Introduction

In the early stages of developing a connected device, the process invariably includes choosing the right processor (MCU or CPU?) and applicable transport(s) (Bluetooth? Wi-Fi? LoRa?). When cost is critical and/or batteries are required, microcontrollers (MCUs) are often the default. It is indisputable that the IoT will consist mostly of these constrained, MCU-class devices. Connectivity options with MCUs are typically bundled with a proprietary software development kit (SDK).

These Software Development Kits (SDKs) have a number of advantages:

- Pre-certified stacks support the hardware configuration, reducing development costs;
- Basic documentation, bundled with the hardware, makes it easy to get started.

However, when choosing a long-term software architecture, relying on the MCU vendor SDK for critical functionality such as networking stacks or software upgrade has some significant drawbacks:

Proprietary SDKs offer low cost and a quick start.

Open source software such as Apache Mynewt and NimBLE offer a better alternative for the long-term.

- **Long-term Support:** Because vendor SDKs are all different, application code portability across hardware is limited to the product line from the corresponding vendor. Cost considerations or inertia often results in de facto lock-in to the chip vendor;
- **Lack of Control**: The core components of the software stack are typically available as pre-compiled binary software, thereby limiting the flexibility or customization possible;
- **Chip and MCU Architecture Dependency**: SDKs and libraries work on only a particular vendor's chips. And, yet, there are restrictions on modification, sharing, and redistribution even in cases where proprietary source code is purchased. And beyond vendor binaries, many distributions take this further by constraining your development only to certain processor cores (e.g., the ARM Cortex-M);
- **Optimized for Prototyping Only**: Many SDKs are optimized for building a prototype expediently and debugging only through a debugger on the workbench. What's missing are the interfaces required to deploy a scalable product that can be managed and maintained remotely for years in production environments in a cost-effective manner.

This whitepaper explores how an open source Bluetooth Low Energy stack under the Apache 2.0 software license removes these concerns and facilitates differentiation in a product--without bloating the budget.

Benefits of source code

Access to open source code unlocks the ability to debug, fix, and enhance your implementation--independent of chip vendor support. "Scratch your own itch" is a given with open source. Not only can the developers better understand how the code works, but they can debug and harden the code by tracing, setting breakpoints, choosing appropriate compiler options, etc. In contrast with a fixed-size binary, developers can recover memory and storage resources by removing functions and data not needed by the application. Conversely, developers can also optimize the application for performance instead of sizing. In short, source code gives developers access and control.

Below are just a handful of examples of real-life problems that product developers have tackled successfully with source code in their hands.

RAM and Flash: More Application Space

Open source software offers the freedom to modify code and freedom from a single vendor.

The result is solution flexibility.

A device manufacturer opted for a very constrained BLE SoC but wanted to add an application protocol to control the device. The Apache Mynewt BLE stack "NimBLE" allowed Runtime to squeeze both the host and controller subsystems into 128 KB of flash and 16 KB of RAM onboard the chosen microcontroller. Enough space was left in that memory configuration to add a complete application layer framework from the Open Connectivity Foundation (OIC 1.1) atop BLE to message and manage the devices in a standards-based way.

Better Debugging

A customer has tight timing requirements and the interrupt handler in the binary RTOS from the hardware vendor is missing interrupts causing synchronization errors and delay. Without the ability to disable/enable interrupts in affected parts of the code, debugging becomes impossible. With open source, interrupt problems are easy to find and fix.

More Throughput

A customer had multiple downstream BLE-connected sensors and required an aggregated throughput beyond what would be allowed by a central over a single connection with each sensor. Runtime helped the customer develop iOS and Android apps acting as the central that created 5 concurrent connections, spoofing and cycling through 5 different device IDs, essentially quintupling throughput. As a result, the customer avoided the additional cost of moving to Bluetooth Classic for the aggregation point.

Quality-of-Service
A body sensor network manufacturer wanted to connect multiple peripherals to a central and guarantee both high throughput and low latency for application data packets of a fixed size. The scheduler of the central had to allocate fixed time slots to the peripherals and guarantee those fixed time slots were available. Enabling faster connection intervals would ensure both a high effective data transfer rate and low latency if the peripherals were scheduled in quick succession. Runtime implemented this streaming service on the central using Apache Mynewt's NimBLE controller with configurable data packet size, number of peripherals, and throughput limits.

The solution allowed the customer to capture statistics such as the number of connections, total packets received, total bytes received, the packets/sec and bytes/sec over the last 10-second interval, etc. Quality of Service (QoS) was provided by guaranteeing that stale data got discarded and by making channel maps configurable for each active connection to allow the customer to avoid bad channels.

Open source software such as Apache Mynewt and NimBLE have pushed the limits of performance and configurability.

The gain extends beyond the community to commercial entities as well.

Frame Interleaving
A customer wanted a peripheral device to advertise different advertisement data and parameters to different centrals in BLE 4.2. Flexibility was desired in terms of the number of each type of advertisement packet to interleave during transmit. Since Apache Mynewt's NimBLE controller allowed multiple advertising instances in BLE 4.2, Runtime leveraged it to add system configuration variables to enable them and configure how many and how often to send each advertisement. The Apache Mynewt controller was able to advertise while a scan or connect operation was in progress. It was able to accommodate the multiple advertisements in between any scheduled event (e.g., connection event).

Increased Concurrent Connections for Location/Proximity Services
A customer needed as many concurrent connections as possible for rapid presence detection. Runtime tested the required data exchange on up to 32 concurrent connections using the Apache Mynewt stack. A single setting was introduced to conveniently specify the maximum number of connections, allowing the customer and others in the community to test beyond 32 connections. Such open source

activity spurred some vendors to increase the maximum number of connections allowed in their binaries (!), but there is still a gap--any flexibility to try out connections beyond the hardcoded limit is missing. A video of the demo with Red Bear Lab can be viewed here.

Additional benefits of Apache Mynewt and NimBLE

Apache Mynewt's highly modular design makes its NimBLE stack decomposable. Just as the secure bootloader in Apache Mynewt was evolved into the open source MCUboot project that offers an OS- and hardware-independent secure bootloader for 32-bit MCUs, the NimBLE stack can be used as a component by other operating systems. Ports are already underway to other RTOSes, increasing the ease of interoperability because of a shared core code base.

Simply offering source code to the developer without a more user-friendly approach does not in-and-of-itself promote fast understanding and development. Exposing a highly granular set of system configuration variables as build time options is one way of tackling the complexity of implementing custom behavior in an easily testable way. The Apache Mynewt SDK does this. Moreover, it allows these system configuration variables to be defined in any package or library, and allows overrides of assigned values based on the priority of the type of the package. It makes it easier for the contributor to introduce new, granular settings via low-level APIs while keeping it simple for the application developer to use overrides to tweak the settings at a higher level. It is an example of an open source, flexible BLE SDK that empowers the product developer far more than just an RTOS kernel bundled with a BLE connectivity stack. From enabling detailed debug settings in simulator mode to enabling full transceiver statistics for in-depth behavior analysis (both pre- and post-deployment), the non-default options are plentiful and easy to exploit.

A permissive license encourages adoption of open source software.

An active community improves the quality and security of open source software.

Permissive License and Community Model

Good open source offers an effective way to reach the maximum audience at the minimum price. The goal is to encourage adoption, and a permissive license such as the Apache License 2.0 does exactly that. It is conducive to commercial development and proprietary redistribution without dramatic changes in the business model of the end product.

The Apache Mynewt BLE stack has been able to build a rich ecosystem with contributions from individuals working on different products. The resulting codebase is robust against a wide variety of test cases, benefitting all in the community.

Active usage and testing of the code by the community also aids in qualification. Including the qualification tests and results against the BT SIG Profile Test Suite (PTS) in the open source repository readies the host subsystem for certification. The user can mix and match this against any qualified controller subsystem from any hardware vendor. If the open source project supports a particular BLE radio at the controller level, as does Apache Mynewt, the community would be assured by a qualified controller system. End product qualification of devices based on that hardware would be simplified for the product vendor. Additionally, commercial support providers such as Runtime can simplify the process further for the device manufacturer through certification services.

In a thriving, active community, open source reinforces sound security practices by involving many people who expose and fix bugs quickly, share concrete examples of secure code, and even conduct unsolicited security reviews! The code is auditable by anyone – inside or outside the community - and there is no security by obscurity.

Conclusion

An open source OS gives customers control over their software architecture and provides a sustainable software platform that is independent of the chosen hardware. Examples from real life have proven the power and flexibility of the open source Apache Mynewt OS coupled with the NimBLE BLE stack.

About Runtime:

Runtime provides cloud-based management of connected devices as part of an end-to-end IoT platform for consumer, medical, commercial, and industrial IoT applications. Runtime supports Apache Mynewt, an open source embedded OS optimized for networking and remote management of constrained, microcontroller-based systems. Apache Mynewt includes the world's first open source, controller-level BLE 4.2, Bluetooth 5, and Bluetooth Mesh stacks for MCUs and a highly efficient OIC 1.1 application framework for managing the connected devices.