February 2022

SIMULATION OF PERSONAL AREA NETWORK DEVICES

Erwin Jansen

Myles Watson

Paris Hsu

SIMULATION OF PERSONAL AREA NETWORK DEVICES

ABSTRACT

Developers of applications (which may be referred to as "apps") for various devices (including smartphones, smartwatches, smart glasses, extended reality – XR – headsets, virtual reality – VR – headsets, augmented reality – AR – headsets, gaming systems, streaming gaming systems, desktop computers, laptop computers, tablet computers, televisions, etc.) may often interface with personal area network (PAN) devices via potentially complex PAN protocols. Example PAN protocols include Bluetooth® protocols, ultra wideband (UWB) protocols, Institute of Electrical and Electronics Engineers (IEEE) 802.15 wireless PAN protocols, Z-Wave® protocols, Zigbee® protocols, etc. Rather than develop and test such apps with respect to physical PAN devices, the developers may load a simulated PAN device that is implemented as a remote procedure call (RPC, such as a general RPC denoted as "gRPC") transport that effectively emulates PAN services for delivery of emulated PAN data between the PAN device and an emulated operating system (OS). The emulated PAN data may be manipulated by the developer to accommodate development and/or testing of the app, which may effectively eliminate a primary difficulty when interfacing with physical PAN devices. The emulated PAN data may be packaged using open source protocols, such as a protocol buffers (which are often referred to as "protobufs").

DETAILED DESCRIPTION

Personal area network (PAN) wireless devices are increasingly being offered to compliment primary computing devices, such as smartphones, smartwatches, smart glasses, extended reality – XR – headsets, virtual reality – VR – headsets, augmented reality – AR – headsets, gaming systems, streaming gaming systems, desktop computers, laptop computers, tablet computers, televisions, etc. For example, rather than include an analogue audio connector (or, in other words, a headphone jack), smartphones may remove the headphone jack in favor of wireless PAN headphones to potentially improve dust and water resistance ratings for the smartphones. As another example, smartphones may interface with wireless PAN heart rate monitors to avoid cables that may otherwise get in the way while the user exercises.

The growing number of PAN wireless devices and the increasing use of such PAN wireless devices by users has resulted in developers attempting to accommodate this user preference in applications (which may be referred to as "apps") under development. However, as the number of PAN wireless devices increases, both across vendors and within vendors themselves, developers of apps may experience issues with PAN wireless device compatibility, which may hinder app development and testing. To illustrate, consider a PAN wireless heart rate monitor device in which there may be multiple different vendors (and possibly multiple different versions) of PAN wireless heart rate monitors. In order to validate an app under development that supports PAN wireless heart rate monitors, the developer may have to physically test each vendor's (and each individual version from each vendor's) PAN wireless heart rate monitor. Even with such physical testing of PAN wireless heart rate monitors, the developer may find it difficult to validate all test cases given that the developer may not be able to directly control or manipulate the PAN wireless heart rate monitor via a test bench or similar developer environment.

In addition, implementation of PAN protocols over which PAN wireless heart rate monitors establish communication sessions with the user device, such as a smartphone (or any one or more of the primary computing devices listed above), may vary due to the complexity of such PAN protocols. For example, one PAN protocol referred to as Bluetooth® is a large standardized protocol having multiple different profiles and services spanning a number of different use cases and settings. The PAN wireless heart rate monitors may differ slightly in the individual implementation of such PAN protocols, which may further frustrate physical testing of PAN wireless heart rate monitors that detracts from app development and testing. In other words, the developer is not concerned with the individual PAN protocol implementation as that impacts how the PAN wireless heart rate monitor connects to the primary computing device (and likely will be addressed through firmware or other updates at some later time). The developer is concerned with how the PAN wireless heart rate monitor provides the underlying data that is used by the app under development.

While described with respect to a Bluetooth® heart rate monitor for the remainder of this disclosure, it should be understood that the following description may apply to any type of PAN wireless device, including PAN headphones, PAN headsets, PAN speakers, PAN smartwatches, PAN smart glasses, PAN blood oxygen sensors, PAN glucose monitors, PAN sensors (e.g.,

proximity sensors), PAN smart home devices (such as PAN light switches, PAN blinds, PAN appliances, etc.), and the like. Moreover, while described with respect to Bluetooth® below, the following description may apply to any type of PAN protocol, such as ultra wideband (UWB) protocols, Institute of Electrical and Electronics Engineers (IEEE) 802.15 wireless PAN protocols, Z-Wave® protocols, Zigbee® protocols, etc.

In the example of FIG. 1 below, a computing device 100 may support a developer environment executing an emulator 102 and a simulated PAN device 104 (which as noted above is assumed to be a Bluetooth® wireless heart rate monitor for purposes of this example). Emulator 102 may represent an execution environment that supports emulation of a potentially different operating system than a primary operating system (which is not shown in the example of FIG. 1) executed by computing device 100. Emulator 102 may also emulate hardware for a particular device that is different from hardware of computing device 100.

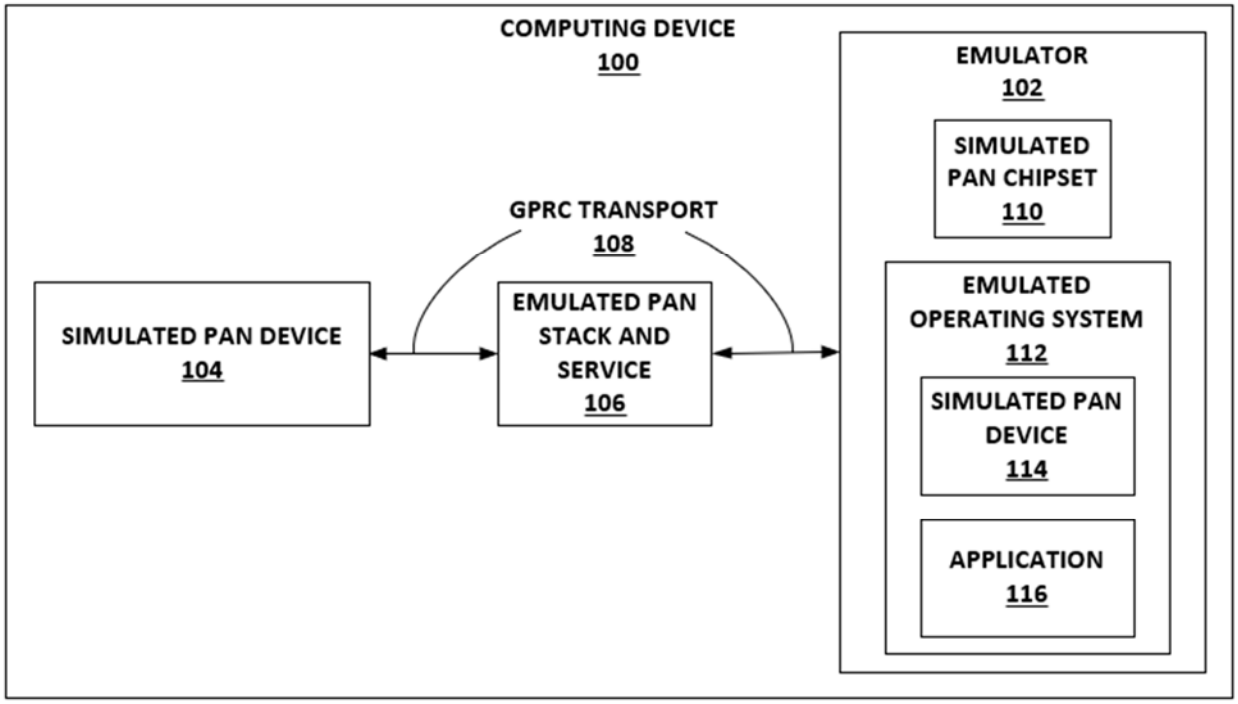

FIG. 1

As shown in the example of FIG. 1, computing device 100 may simulate communication between emulator 102 and simulated PAN device 104 (which may also be referred to as Bluetooth® (BT) heart rate monitor 104) using an emulated PAN stack and service 106 (which may also be referred to as an emulated Bluetooth® (BT) stack and service 106). This simulated

communication between BT heart rate monitor 104 and emulated BT stack and service 106 and between emulated BT stack and service 106 and emulator 102 may occur via a generated remote procedure call (gRPC) transport 108. Emulated BT stack and service 106 may support BT services, such as a generic access protocol (GAP) and generic attribute profile (which may be denoted as "GATT"). While described with respect to the open-source gRPC, any form of RPC may be used to establish transport 108.

Emulator 102 may support, as noted above, software emulation of hardware and software (when not supported by the underlying processor architecture of computing device 100). In the example of FIG. 1, emulator 102 performs software emulation to support execution of simulated PAN chipset 110 (which may also be referred to as simulated Bluetooth® (BT) chipset 110) and emulated operating system 112.

Simulated BT chipset 110 may simulate BT chipsets for the physical layers of the BT stack that implement baseband controllers (including link controller, link manager, host controller interface – HCI, etc.). Emulated operating system (OS) 112 may represent an operating system that is not natively supported by the underlying physical hardware of computing device 100 (due to a different processor architecture, such as x86 compared to advanced RISC machines or ARM, where RISC refers to a reduced instruction set computer).

Emulated OS 112 may support an application space in which an application 116 (which may be referred to as app 116) may execute to facilitate development and testing. The developer may load compiled program code and install app 116 within the application space supported by emulated OS 112. Emulated OS 112 may then execute app 116 to support testing.

In order to test BT connectivity to simulated BT heart rate monitor 104 for testing of app 116, computing device 100 may support execution as shown in the example of FIG. 2. First, simulated BT heart rate monitor 104 may register simulated BT heart rate monitor 104 (200) directly with emulator 102 to initiate configuration and execution of emulated BT stack and service 106. In this sense, emulator 102 may launch emulated BT stack and service 106 as a process within emulator 102 (202) to simulate a BT stack on behalf of emulated BT heart rate monitor 104.

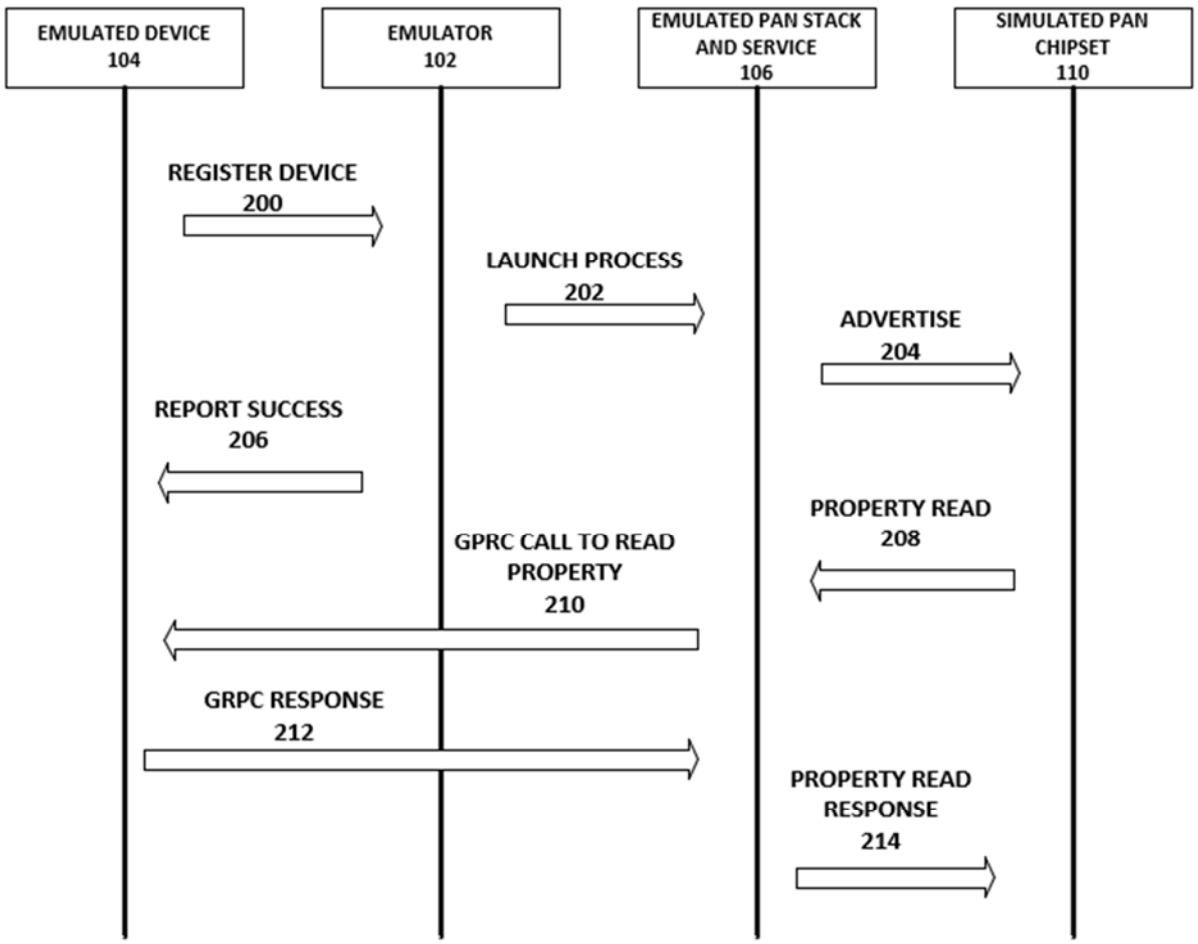

FIG. 2

Once emulated BT stack and service 106 is launched, BT stack and service 106 may implement the BT stack and services (e.g., GAP and GATT) to advertise emulated BT heart rate monitor 104 (204) as available for establishing a BT connection (or, in other words, is available to pair with emulated OS 112) via simulated chipset 110. In this sense, emulator 102 executes simulated BT chipset 110 on behalf of emulated OS 112, which proceeds to establish the BT connection between simulated BT heart rate monitor 104 and OS 112. Upon registering simulated BT heart rate monitor 104, emulated OS 112 may effectively have a BT service associated with simulated BT heart rate monitor 104, which is shown above in the example of FIG. 1 as simulated PAN device 114.

In any event, emulator 102 may report successful registration with emulator 102 and pairing with emulated OS 112 back to emulated BT heart rate monitor 104 (206). At this point,

emulated BT heart rate monitor 104 functions as a simple state machine that responds to reading, writing, and streaming of properties. As such, app 116 may interface with emulated OS 112 to issue, as an example, requests to read a property exposed by simulated BT heart rate monitor 104 (such as a current_heart rate property). Emulated OS 112 may interface with simulated BT chipset 110 (via a BT driver) to issue the property read.

Simulated BT chipset 110 may formulate the property read in accordance with BT protocols to obtain a BT packet. Simulated BT chipset 110 may then transmit the BT packet via gRPC transport 108, effectively transmitting the property read to simulated BT stack and services 106 (208). In some instances, the gRPC transport 108 utilizes an open source data serialization structure referred to as protocol buffers (which are often referred to as "protobufs"). In these instances, simulated BT chipset 110 may store the BT packet as one or more protobufs and then transmit these protobufs via gRPC transport 108 to simulated BT stack and services 106.

Simulated BT stack and services 106 may receive the BT packet as the one or more protobufs via gRPC transport 108 and reassemble the BT packet from the protobufs. Simulated BT stack and services 106 may process the BT packet to identify the property read. Simulated BT stack and service 106 may then generate a gRPC call to read the identified property, transmitting the gRPC call to simulated BT heart rate monitor 104 (210). Simulated BT heart rate monitor 104 may receive the gRPC call and then perform a lookup on the property identified in the gRPC call (where such lookup is completely programmable and in control and manipulation by the developer) to obtain the property value.

Simulated BT heart rate monitor 104 may then generate a gRPC response that includes the property value and transmit the gRPC response via gRPC transport 108 to emulated BT stack and services 106 (212). Emulated BT stack and services 106 may then process the property value to formulate a BT packet that specifies the property value in accordance with BT protocols/services. Emulated BT stack and services 106 may then store the BT packet as one or more protobufs and transmit the protobufs via gRPC transport 108 back to simulated BT chipset 110, effectively transmitting the read response to simulated BT chipset 110 (214).

Simulated BT chipset 110 may receive the protobufs via gRPC transport 108 and unpack the BT packet from the protobufs, processing the BT packet to retrieve the property value. Simulated BT chipset 110 may store the property value in memory and issue an interrupt to

emulated OS 112 to alert emulated OS 112 that the property value corresponding to the property read request is available. Emulated OS 112 may provide the property value to app 116 (e.g., pass a pointer to the location in memory at which the property value is stored). App 116 may read the property value from memory, processing the property value for various purposes specific to app 116 (e.g., display, further computation, etc.).

In this way, a developer may logically implement data communication via BT with an emulated BT device without having to worry about BT implementation specific details of each physical BT device. The developer may then focus on data compatibility between emulated BT device 104 and app 116 while also having extensive control of the data that is communicated between emulated BT device 104 and app 116. In addition, by enabling such extensive control of the data that is communicated between emulated BT device 104 and app 116, the developer may automate testing, potentially enabling extensive test beds that can rapidly iterate between a large number of different (be it vendor or version within any given vendor) BT devices.

It is noted that the techniques of this disclosure may be combined with any other suitable technique or combination of techniques. As one example, the techniques of this disclosure may be combined with the techniques described in WIPO Application No. 2003081428A2. In an additional example, the techniques of this disclosure may be combined with the techniques of "BT-Sim:A Bluetooth simulator". "BT-Sim: A Bluetooth simulator" may be found at Sourceforge at the following URL: http://btsim.sourceforge.net/.