CHALLENGES TO BUILDING BLUETOOTH-BASED SENSING SOLUTIONS

Chieh-Yih Wan
Intel Corporation
chieh-yih.wan@intel.com

Sai Prasad
Intel Corporation
sai.prasad@intel.com

ABSTRACT

Emerging context aware applications call for new networking technologies to enable rapid development of integrated solutions that gather, process and store context from a diverse set of sensors. We examine Bluetooth in the context of enabling emerging classes of context aware applications, such as healthcare, fitness, gaming, etc., using off-the-shelf (OTS) products. While Bluetooth is widely used today, its applicability to this new class of applications is not widely understood and applications that use Bluetooth could suffer from inconsistent usages and poor performance as a result. We investigate and report the challenges of implementing solutions that use software and OTS products based on existing Bluetooth standards. We also present performance analysis through experimentations to highlight some of the issues discussed in the paper. Based on our experience from building Bluetooth based sensing solutions, we make informed recommendations for modifications in the Bluetooth standard and highlight areas where new standards are required.

1. INTRODUCTION

A new class of applications is rapidly emerging around gathering, processing and storing of context data from a diverse set of sensors located on-body, on the host platform and in the environment. Context data can be used to answer questions such as “where am I”, “what am I doing” and “who’s around me.” The high level system architecture of these applications often includes an aggregator associated with a given person which collects data related to that person from various sensors. Typically, the aggregator would be implemented on a mobile device, such as a PDA, Smartphone, or Mobile Internet Device (MID). Unlike most conventional BSN applications that rely on custom-built hardware, context-aware applications can benefit from the proliferation of Bluetooth-based sensing devices available in the market today that support on-the-go lifestyles for mobile users.

We examine Bluetooth in the context of enabling emerging classes of context aware applications, e.g., healthcare, fitness, gaming, etc., using Off-The-Shelf (OTS) products available in the market today. While Bluetooth is widely used today, it was primarily designed as a cable replacement, and its implementation and performance issues in context-aware applications are not widely understood. The Bluetooth specifications have left several design issues open to implementation relative to its use as a networking technology [22]. Most prior work in Bluetooth research has been focused on developing efficient scatternet formation strategies to support ad hoc networks [25] [26] [22] or personal area network (PAN) [27]. Our work is distinct from prior research in that we focus on the impact of Bluetooth issues on the application and user experience rather than the network.

The requirements and usage of Bluetooth in emerging context-aware applications is different from typical usages where Bluetooth is used to connect to peripheral devices. The aggregator needs to interact with a constellation of highly diversified sensors, acquire, aggregate and correlate the context data from these different sources to make sense of the environment and situation of the user. The dynamic nature of a constantly changing user environment imposes additional user and application challenges around interacting with sensors over Bluetooth. We have discovered a number of serious issues that are not addressed in current implementations of a number of OTS Bluetooth sensors commonly used in the aforementioned application scenarios. We study the performance impact of some of these issues through experimentations to highlight the challenges.

The purpose of this paper is two-fold:

1) To suggest the need for modifications in the Bluetooth standard and implementations, including cases where existing standard features have not been adopted.
2) To provide useful guidelines to system designers for developing Bluetooth-based context aware applications.

Note that recently there have been a number of new additions [2] [3] to the Bluetooth standard that are beginning to address some of the issues we have identified. In this paper, we consider several cases: 1) Issues that are not addressed in any version of the Bluetooth standard, either because the standard does not consider the issue or the issue is out of scope; 2) Issues that are addressed in the most recent Bluetooth standard, which has not been widely adopted (if at all) by device manufacturers; 3) Issues that can be addressed by other standards, if used in conjunction with Bluetooth.

The rest of this paper is organized as follows. Section 2 presents a matrix of Bluetooth features required for emerging context aware applications and highlights the high level challenges. Section 3 describes in detail the challenges that are related to standards. Section 4 discusses implementation challenges using existing tools (SDKs, libraries). Section 5 presents experimental

* Other names and brands may be claimed as the property of others.

Digital Object Identifier: 10.4108/ICST.BODYNETS2009.6022
http://dx.doi.org/10.4108/ICST.BODYNETS2009.6022

evaluation on some of the challenges described in earlier sections, followed by a conclusion in Section 6.

2. BLUETOOTH AND CONTEXT-AWARE APPLICATIONS

Bluetooth was originally designed as a cable replacement to connect devices such as mobile phone handsets, headsets and portable computers [4]. Today, due to its pervasiveness, Bluetooth has been widely used for short-range wireless communication between a host of electronic devices. The explosive growth in Bluetooth based products in the past few years has extended the technology into new application areas, such as connecting mission critical medical sensors[1]. The number of new applications and new usages simply outpaces the development of Bluetooth standards and implementations.

We present below, a taxonomy of emerging applications centered around context collection and aggregation.

Table 1 presents a matrix of the characteristics of a class of emerging applications and the features required to implement them. The table also highlights the challenges in implementing these features either due to gaps in existing Bluetooth standards or issues in the application space. The feature characteristics presented in the table are color-coded to indicate the degree to which they are supported in existing Bluetooth standards and implementations.

The feature characteristics of the emerging applications are:

- **Fast Association**: Rapidly pair a host with a device seamlessly, reliably, without user intervention.
- **Security:** Protect sensor data streams from eavesdropping and tampering through authentication and encryption.
- **Audio**: Support hands-free voice interactions.
- **Streaming**: Support connections that remain open for an indefinite period of time to stream sensor data continuously.
- **Periodic/Batch transmission**: Reliable delivery of an amount of payload that is known a priori.
- **Auto Reconnect**: Reconnect seamlessly if the connection to sensors is lost.
- **Mixed Master / Slave Mode**: Support simultaneous connections to multiple sensors operating in either master or slave mode.

The details of these challenges are discussed in later sections of this paper. In the table, we list the features that would be required to support applications in different usage scenarios, based on typical usages for which the applications are designed. For example, in the Gaming usage scenario, the sensors are often dedicated and owned by the user for a relatively long period of time. The requirement for Fast Association between the host and the sensors is optional because the associations can be setup once and they remain active permanently or semi-permanently. On the contrary, the sensors used for Fitness in a Fitness Center will need to be shared among multiple users, and Fast Association is thus mandatory in this usage scenario.

[1] Note that devices, sensors are used interchangeably in this paper to refer to the source devices that generate data, and host refer to the aggregator or receiver of data from the sensors.

		■	■	▩	▩	▨	▨	□
Applications	Typical Sensors	Fast Association	Auto Reconnect	Audio	Security	Mixed Master Slave Mode	Streaming	Periodic/Batch Transmission
Emergency Services	BP, EKG, SpO2, EtCO2	✓	✓	✓	✓	✓	✓	✓
Hospital Care	BP, EKG, SpO2, EtCO2	O	O	O	✓	✓	✓	✓
Elder Care	BP, Weight, Blood sugar		✓	✓	✓			✓
Disease Prevention	BP, Activity, Weight, Blood sugar		✓	O	✓	✓	✓	✓
Fitness	Pedometer, activity, fitness equipment, weight	✓	O	O	O	O	✓	✓
Personal Wellness	BP, Weight, Temperature		O	O	O			✓
Gaming	Accelerometer, HID,	O	O	O	O		✓	
Peripheral Connectivity	Mouse, Printer, keyboard, joy, headset, earpiece		O	O	O	✓	✓	

✓ Feature is required O Feature is optional

■ No support in standard
▩ Supported by standard, limited by BT stack/HW
▨ Supported by standard, challenges in the application space
□ Supported by standard, no major challenges

Table 1: Feature Matrix for Emerging Context Aware Applications

2.1 Context Acquisition Building Blocks

Context aware applications acquire data from a variety of sources such as sensors and use an inference process to deduce context. The acquired data may be rendered, stored locally, or transmitted to backend servers for further processing. Figure 1 shows the functional blocks required to acquire context from sensors in a typical context aggregation framework. Bluetooth is shown as the primary sensor interface, though in reality, sensor interfaces for multiple wired/wireless technologies could be included. Several context aggregation architectures have been developed. miTag [23] and CodeBlue [24] are two example distributed context aggregation systems that utilize IEEE 802.15.4 radio technology for medical sensor networks applications while our work focuses on Bluetooth.

We highlight some of the challenges to implementing these functional blocks either due to gaps in existing Bluetooth standards or due to lack of standards in certain areas such as sensor data protocols/formats.

Sensors: Sensor data formats are often unique and proprietary to each sensor (vendor specific). Depending on the sensor type, the data is either scalar (e.g. Blood Pressure, weight) or vector (e.g. waveform data from electrocardiogram (EKG), accelerometer). The Bluetooth pairing process establishes a trusted relationship between the host and the sensor. Section 3.1.1 describes the pairing process and issues.

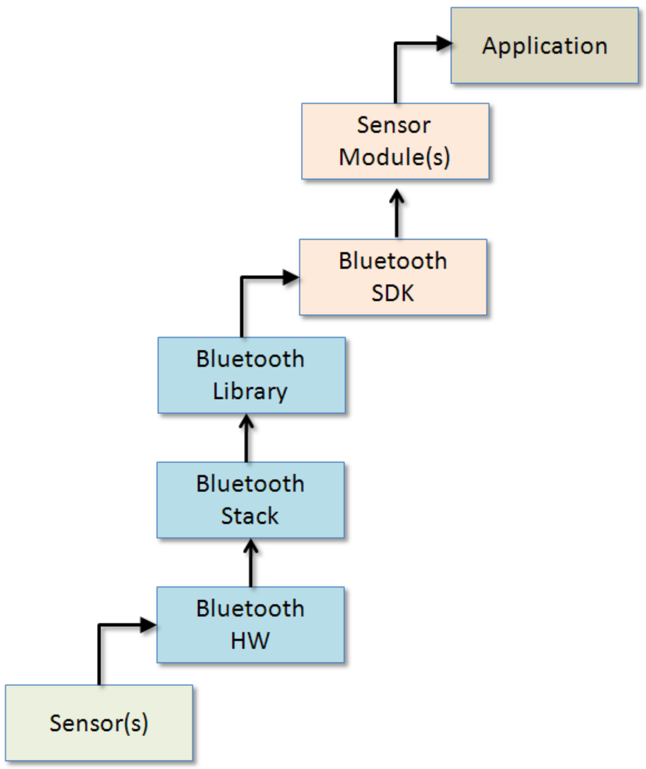

Figure 1: Context Acquisition Building Blocks

Bluetooth Hardware: The Bluetooth hardware (chipset) implements radio and baseband layers that comprise the lower layers of the Bluetooth communication protocol. Mobile platforms today are equipped with Bluetooth hardware and the integration is usually over USB. Some vendors provide integrated Bluetooth hardware solutions with the higher layers such as L2CAP, SDP, RFCOMM and all the profiles [1] embedded in the hardware itself. These are widely used in sensor devices as they can be easily integrated, typically through a serial interface. Integrated hardware solutions make it very difficult to update bugs or errata in the Bluetooth standard. For example, there are significant changes between version 1.0b and 1.1 specifically to improve reliability and interoperability. Devices that use an older version of these integrated chipsets suffer from interoperability issues, as described in Section 3.1.

Bluetooth Stack: The Bluetooth stack is the set of drivers that communicate directly with the Bluetooth hardware. There are two widely used Bluetooth stacks: the Microsoft stack [10] that is bundled with the Windows operating system and the WIDCOMM stack [11][14] from Broadcom Corporation. The choice of stack depends on the Bluetooth chipset installed in the host platform. Section 4.1 presents the challenges related to Bluetooth stacks.

Bluetooth Library: A Bluetooth library is supplied by the Bluetooth stack vendor, allowing applications to access services from vendor's Bluetooth stack through an API. Bluetooth libraries are tightly integrated with Bluetooth stacks, and a Bluetooth library from a vendor will work only with the Bluetooth stack from that vendor. Section 4.1.2 presents the related issues.

Bluetooth SDK: Commercially available Bluetooth SDKs provide application portability across platforms and across Bluetooth libraries/stacks from different vendors. SDKs offer features such as support for the .NET frameworks, portability across platforms (Windows and CE), and portability across vendor libraries. Section 4.2.1 discusses in detail some of the SDK related issues.

Sensor Modules: Most sensors send data as 'frames'. The frame format for each sensor is proprietary and defined by the sensor manufacturer. Sensor Modules implement the logic to parse the sensor data stream according to specifications from the sensor manufacturer. Since individual sensor vendors chose the frame format and contents, features required to perform error handling, such as sequence numbers, frame lengths, and robust start and stop codes, are not always present. Section 3.2 presents a deeper dive into the lack of sensor data standards issues.

The next two sections detail the challenges in each category.

3. STANDARDS BASED CHALLENGES

A number of standards related challenges arise in implementing context aware applications based on Bluetooth technology. The challenges fall into two broad categories: those related to the Bluetooth standard and those due to lack of standards to represent sensor data.

The Bluetooth standard has been through a number of major revisions [4] – from 1.0, 1.0b, 1.1, 1.2, 2.0 to 2.1. Most revisions are incremental improvements over earlier revisions and are backward compatible. However, many of the improved features in 1.1 such as, low-power modes, security procedures and role switching, are not backward compatible with 1.0b [15]. This section describes challenges related directly to the Bluetooth standard and data formatting standards.

3.1 Bluetooth Standards

Bluetooth is an actively evolving standard. Most of the OTS products in the market today are still based on the older versions 1.1 and 1.2. The latest version 2.1 [1] (released July 2007) and the recently approved (late June 2008) Health Device Profile (HDP) [2] and Multi-Channel Adaptation Protocol (MCAP) [3] are beginning to address some of the issues mentioned in this section. For example, MCAP supports fast reconnection (see Section 3.1.3) and defines a time synchronization protocol for data time-stamping (see Section 3.1.4) in the sensors. However, issues such as pairing between a sensor and multiple hosts (see Section 3.1.1.2) are not addressed by the latest version of the Bluetooth standard. Furthermore, products supporting these new additions do not exist even today.

The following sub-sections describe some of the major challenges related to the current Bluetooth standard.

3.1.1 Fast Association

Pairing is a process by which Bluetooth devices establish a trusted relationship with each other. User interaction is required only during pairing where the user may have to enter the passkey manually. Bonding between a host and the peripherals or sensors can be permanent or temporary depending on the usage. Permanent bonding applies to dedicated or long-term host-device associations, e.g., keyboards, mice, or medical devices such as weighing scales and BP monitors used for home care. Temporary bonding is used in usages where sensors may be frequently swapped or paired with multiple hosts on a daily basis and the associations between host-device pairs are short-lived. Some devices used in Fitness such as treadmills, fall into this category, and they are shared among multiple users. The key requirement here is for the pairing to be quick, easy and seamless. Prompting the user for a security pin every time the device is used can be annoying or other constraints such as lack of an input device can make it impractical. The next subsections discuss the challenges to make temporary pairing quick and easy, and suggest approaches to address them.

3.1.1.1 Device Identification

To pair a device with a host, it has to be identified to the host by its address. For pairing to be quick, the device must be identified to the host with minimal user interaction. Woodings, etc., [21] propose a mechanism to accelerate Bluetooth connection establishment time between two devices by using an IrDA connection to exchange device discovery information required to establish the Bluetooth connection. The approach requires both devices to be equipped with IrDA capability, a requirement that likely cannot be met considering most OTS sensors do not support IrDA. The Bluetooth standard 2.1 [1] lists several association models that may be appropriate. In the *Out of Band* association model, the parameters required for pairing are exchanged through an out-of-band channel. For example, a barcode on the device encoded with the address and pin number can be scanned to initiate pairing through the host application. The *Just Works* association model uses a numeric passkey that is never displayed. The user is merely asked to accept the incoming connection on the host. These two association models use single-touch processes for pairing. It should be noted that the *Out of Band* and *Just Works* association models are not a part of the Bluetooth standard prior to 2.0.

A large number of Bluetooth devices in the market today are based on versions 1.1 or 1.2 of the standard. They use the *Passkey Entry* association model that requires the user to enter a numeric key on the host. It is important for device manufactures to adopt *Fast Association* models for sensors that will be used in critical applications where host-device associations change frequently.

3.1.1.2 Pairing with Multiple Hosts

When a device is paired with a host, both the device and the host store the link keys for the association. The stored keys are subsequently used when the two connect, speeding up the connection process. However, a given device typically limits the number of link keys that can be stored, due to resource constraints. Some devices allow pairing with only one host at a time and once they are bonded, they stop responding to device inquiries from other hosts, i.e., they become non-discoverable. Others store link associations with more than one host. The user has to go through the pairing process, potentially with different steps, under different conditions. For instance:

1. When a new association is needed between a device and a host, the user has to go through the normal pairing process.
2. When the device runs out of room to store additional link associations, some previous associations on the device have to be removed to make room for the new ones before going through the normal pairing process. The removal of previous association(s) on the device can either occur automatically when a new association is added, or may require the user to press one or more buttons on the device for an extended period of time. In both cases, when the previously bonded hosts need to talk to the device, the user will need to go through the pairing process again because their original associations have been removed as a result.

Because these two cases require different procedures, and it is difficult for the user to know whether or not the device can accept additional link keys, user confusion and frustration can result.

Pairing may also be challenging for devices based on legacy versions of the Bluetooth standard (pre 1.1). If such a device is already bonded to host A, the user pairs it with host B, and then tries to connect or pair it with host A again, connection or pairing may fail indefinitely. This is because mutual authentication is not enforced in legacy versions of Bluetooth [15]. If an old link key is reused at one end, pairing/connection will fail because the keys do not match. The only workaround is for the user to delete the device from the list of known devices (e.g., via the Windows Bluetooth Manager[2]) on the host and re-try pairing.

3.1.2 Master versus Slave Mode

Bluetooth devices operate in one of two modes: Master or Slave. Communication between Bluetooth devices is based on the Time Division Duplex (TDD) scheme [1] in which a Master provides a common clock and frequency hopping pattern for sharing the physical channel. Slaves synchronize to the Master in time and frequency by following the Master's hopping sequence. The device that initiates the connection (paging) becomes a Master, and the paged device becomes a Slave. Bluetooth-enabled sensors operate either in master mode or slave mode depending on the targeted use cases.

3.1.2.1 Master Mode

Typically, a device that collects a small amount of data sporadically or on an on-demand basis operates in the master mode enabling simpler and more efficient power management. A good example is a blood pressure (BP) monitor device. A BP monitor takes a measurement either when the user presses a button or automatically at regular intervals. By operating in the master mode, the device can save power by keeping the Bluetooth hardware powered off and only power it on to upload data to a host after a measurement has been taken. Note that in this case, the sensor controls the timing for connection assuming that either 1) the host platform is always ready to establish the communication link for the data upload, or 2) the sensor platform has enough memory to store the collected data until the host is available for upload. Note that in the former case, the host platform must support the Bluetooth Scatternet feature to communicate with more than one master sensor simultaneously.

3.1.2.2 Slave Mode

Data streaming devices such as accelerometers and Pulse-Oximetry (SpO2) sensors often operate in slave mode while delegating timing and control of connections to the host platform. This is done for several reasons:

Real-time requirement: The host software must be ready to receive data from the sensor in real-time when data collection starts. An example is a host application displaying EKG or SpO2 photoplethysmographic (PPG) waveform in real-time. As a result, it makes sense for the host to initiate and control the connection to the device.

Storage: Due to higher data rate and continuous data collection processes, resource-limited sensor platforms often do not have enough memory to store the collected data for an extended

[2] This workaround is known to work on Microsoft Windows XP.

amount of time. These sensors often remain dormant (not collecting data) until they are activated by the host.

Sensor configuration: Streaming devices such as EKG sensors often support multiple operation modes and need to be configured with specific parameters before data can be collected. Such parameters include sampling rate, the number of analog-to-digital (ADC) channels and report frequency. The setting of these parameters often has UI or data analysis implications and should be configured by the host software during start-up.

3.1.2.3 Supporting Mixed Mode

Context-aware applications often need to support multiple sensors operating in either mode (master or slave) at the same time. The Bluetooth standard supports mixed mode operations on a device through Scatternet and Master-Slave-Switch [1]. However, depending on the chipset used and HCI firmware version, not all Bluetooth hardware supports Scatternet and role switching features. The issue is mainly due to ambiguities in Bluetooth version 1.0b, as well as incompatibility between 1.0b and later versions [15]. While most PC and handheld devices use hardware that support Bluetooth versions 1.1 or later, a good portion of the medical sensors in the market today still use legacy Bluetooth chipsets. This creates a challenge on the host to support both master and slave sensors at the same time. Until all sensor manufacturers comply with the latest standards, application developers will require workarounds such as using only Slave sensors or working with sensor manufacturers to modify their firmware to operate their sensors in the slave mode.

3.1.2.4 Power Management

Bluetooth supports multiple low power modes, i.e., Standby, Park, Hold and Sniff mode, but only in the connected state. Before a device joins a piconet, the most viable low power mode is to turn off the radio transceiver. For devices that operate in the master mode, turning off the radio works well as they control when connections are made. However, devices that operate in the slave mode wait for connection from a master (e.g., the host platform) and must keep the radio on until they are paged and connection is established. For battery-powered sensors, keeping the radio powered on could drain the battery quickly. A Bluetooth device that is discoverable must regularly perform an inquiry scan before it joins a piconet, and inquiry scan is one of the most power consuming operations. It is therefore common for a slave sensor to turn off inquiry scan and become non-discoverable once it has been paired (or bonded) with another device, and would require an out-of-band method (Section 3.1.1.1) to allow a new host to bond and connect to the sensor without inquiry.

3.1.3 Auto Reconnect

Auto-reconnect minimizes data loss, particularly for sensors that send streaming data such as 3-axis accelerometer and EKG waveforms. A device may disconnect or be unable to send data due to various reasons such as poor signal quality caused by body absorption, RF interference, going out of range, etc., or simply because the device runs out of battery or the user turns off the device. The Bluetooth specification defines a parameter called the *Link Supervision Timeout* (LST) that controls the amount of time an active connection is monitored for missing packets. If no baseband packets are received for this duration (default is 20 seconds), the link is disconnected. However, there are sensors that will stop sending data if they lose connectivity to the host more than a few seconds because of limited buffer space or other resources. When this happens, the aggregator has to disconnect from the sensor and establish a new connection. Packet loss due to reconnection latencies can significantly impact data analysis especially in critical care applications that use data from sensors such as EKG.

The setting of the Link Supervision Timeout should be carefully chosen. If the value is too large, the Bluetooth stack may take a long time before notifying the application that the link has been disconnected, where disconnecting and reconnecting sooner would have reestablished the data streams. If the value is too small, the Bluetooth stack may disconnect from the device prematurely, resulting in unnecessary data loss.

The Health Device Profile [2] that uses the MCAP [3] protocol solves this problem through a reconnect feature that allows an application to quickly reestablish disconnected channels without the overhead associated with a new connection. The application can set the Link Supervision Timeout to a small value, for rapid notification, and use the reconnect feature to quickly reestablish the connection. To enable this feature, the device has to support the new standard and maintain enough state to allow reconnection. However, this feature may not work across sensor reset, e.g., for battery replacement. Section 5.2 presents a comparative analysis of the overheads associated with disconnecting and connecting versus using the reconnect feature.

3.1.4 Data Time Stamping

Accurate time-stamping of sensor data is crucial in context aware applications not only for displaying data, e.g., showing waveform or trends, but more importantly to correlate time-sequence data from different sensors for inference and/or fusion purposes. Sensor devices today often do not have a real-time clock to keep track of wall clock time. Even if they do, clocks drift over time (differently on different devices) and the time differences between different devices makes correlating time-sequence data extremely difficult depending on the drift ranges and accuracy needed. In order to use a common time-reference across different data streams, most applications today timestamp data packets only after they arrive on the host platform. However, time-stamping data on the host has limited accuracy due to a number of factors such as transmission latencies due to retransmissions and processing delays due to system load. In our application, the time jitters introduced by these variables on a relatively powerful PC platform (ThinkPad T61) can skew the data timestamps enough to completely distort waveforms such as the PPG waveform from a pulse-ox sensor, as shown in Section 5.1. As a result, accurate time-stamping of sensor data should be done at the point of acquisition on the sensor device.

To address the aforementioned problems, it is necessary to time synchronize all Bluetooth connected devices within a piconet. Today, a number of time synchronization protocols are available [16] for sensor networks that can achieve micro-second accuracy. However, those protocols often are hardware dependent and/or topology dependent. Given the diversity of the sensor platforms in context aware applications, fine-tuning those protocols on each sensor platform is a significant challenge. On the other hand, each Bluetooth enabled device already has a synchronized clock in the Bluetooth implementation. As previously mentioned, Bluetooth slaves within a piconet synchronize to the Master in real time with high accuracy. It is thus logical to leverage the Bluetooth clock for data time-stamping purposes. Similar approaches have been proposed in [28] [29]. It should be noted that the Bluetooth Clock has no relation to the time of day but just provides the "heart beat" of the Bluetooth transceiver [1]. A higher layer protocol is needed to track and map the "heart beat" to time of day. The recently approved MCAP [3] defines a Clock Synchronization Protocol (CSP) exactly for this purpose. While the CSP is optional in MCAP, and products that support MCAP have yet to be developed, we recommend sensor manufacturers include CSP for all sensors that generate time sensitive data.

3.2 Sensor Data Standards

As mentioned in Section 2.1, currently there are no commonly adopted standards for sensor data protocols and sensor data format. Each sensor manufacturer defines a set of proprietary protocols and data format, and therefore proprietary drivers are needed on the host to format and interpret the data from a specific sensor. Most OTS sensor products today are bundled with demo software to communicate with the sensor and interpret the data. However, it is very difficult and often impossible to reuse the demo code for integrating these sensors with a custom application. Developers often resort to writing custom code for each supported sensor device according to the specifications supplied by the sensor manufacturer. The redundant effort in debugging and unit testing the custom code unavoidably incurs additional development time and cost.

To facilitate genuine interoperability between data sources (sensor) and recipients of device application data (host), there is a need for a standard that defines a base data protocol, command set and device data formats for a diverse set of devices needed in context-aware applications. Such a standard could minimize the need for proprietary drivers and facilitate code reuse from different vendors. The IEEE 11073-20601 Data Exchange Protocol [17] is a standard under development for this purpose. It attempts to solve the issues mentioned above, in the medical space. The standard defines a common framework for making an abstract model of personal health data available in transport independent transfer syntax to support logical connections between medical devices. However, the standard focuses only on medical usages and is still only a draft, unavailable to anyone outside of the working group.

We present one possible approach that can be applied to context aware systems in general. We address the data format issues (Section 3.2.2) by creating template objects that provide a set of common data representations of sensor data for various classes of data types, e.g., EKG waveform type, SpO2 scalar type, 3-Axis acceleration type, etc. We also abstract common sensor management features, e.g., connect/disconnect, sensor errors and system resources representations, etc., through a set of well defined interfaces that encapsulates device specific protocols in a thin layer of middleware. The next subsection focuses on enabling a common sensor management framework that is independent of sensor hardware.

3.2.1 Metadata and Sensor Management

As the diversity and the number of sensors increases, managing, maintaining and troubleshooting sensor related problems could be a daunting task for the average user. It is reasonable to see the high value of generic sensor management software modules that can manage sensors without requiring them to interpret proprietary sensor data. Today, device status is typically embedded in sensor data packets as device specific status bits. System resource information such as battery level and memory usage as well as generic errors such as wire disconnected, are common among most, if not all of the sensor platforms. It is thus beneficial to abstract out the common elements of device information and formalize them into device independent metadata that can be interpreted by generic sensor management software modules.

The metadata idea can be further generalized to eventually support a plug-n-play sensor framework. Analogous to USB devices and usages, when the user connects to a new sensor, the host management software would be able to query and acquire the device information, vendor information, device classes, device data types, etc. using standardized metadata, and load the corresponding sensor driver automatically, perhaps even download the required driver from the Internet. Note that, the newly adopted HDP coupled with the MCAP and the drafted IEEE 11073-20601 Data Exchange Protocol promises a similar solution for medical applications.

3.2.2 Sensor Data

All the sensors in the market today use proprietary data formats. Application developers have to develop custom code to parse data streams in the proprietary formats. Some of the proprietary data formats used by sensors today lack sufficient support to reliably detect missing or corrupted packets through mechanisms such as sequence numbers or CRC.

3.2.3 Common Data Standards

The lack of common standards to represent sensor data makes sensor integration a difficult task. The problem can be addressed at two levels. First, at the sensor level, a standard needs to be adopted to represent the packets in the raw data streams from the sensors. This will enable developers to implement sensor modules that can acquire data from any sensor irrespective of the type of the sensor or the manufacturer. Second, at the application level, there should be a standard way of representing data that is common to all sensors that belong to a specific class. E.g., all BP sensors at the very least send the systolic and diastolic readings. Applications often do not care about the raw data streams but only the context that the data actually represents. A common data representation separates sensing context from sensors (hardware/vendor) and supports application logic that is hardware independent. If the raw data from each sensor were converted to the standards-based format for the corresponding sensor class, applications can be easily extended to handle sensors without any additional code.

4. INTEGRATION CHALLENGES

This section describes the software issues and challenges that developers are likely to encounter when dealing with Bluetooth stacks, Bluetooth libraries and 3rd party Bluetooth SDKs.

4.1 Bluetooth Stack/Library

Bluetooth stacks are tightly coupled with the Bluetooth chipset integrated on the host platform. Depending on the chipset, switching between stacks is sometimes possible though it is not easy.

4.1.1 Support for Hands-free Headsets

The stacks differ in the features they support. The WIDCOMM stack from Broadcom supports more profiles than the Microsoft (MS) stack (e.g., the MS stack does not support the Hands-Free profile under Windows XP, while the WIDCOMM stack does). Switching from the MS stack to the WIDCOMM stack is possible only if the WIDCOMM stack supports the Bluetooth chipset on the host platform, otherwise the Bluetooth hardware itself may need to be replaced. There is also a limitation in the number of Bluetooth headsets that the stacks support due to the number of synchronous channels that are supported by the Bluetooth standard.

4.1.2 Bluetooth Libraries

Bluetooth libraries are supplied by the Bluetooth stack vendor and are typically available as free downloads. They provide functions to access the vendor's Bluetooth stack. The interface exposed by the library is proprietary and each vendor's library will only work with the Bluetooth stack from that vendor. An application that is linked to a specific vendor's Bluetooth library will not work on a host platform that has a Bluetooth stack from another vendor. To make the application portable across vendor libraries, a developer would have to implement a software abstraction layer above the vendor libraries that insulates the application from the vendor-specific interfaces. These factors put an additional burden on the developer, increasing development costs and time-to-market. Commercially available SDKs partially solve this problem as discussed in the next section.

4.2 Bluetooth SDK Libraries

Bluetooth SDK libraries provide a layer of abstraction above the Bluetooth vendor libraries and make it easy to develop applications that are portable across Bluetooth stacks from different vendors. Several commercial Bluetooth SDKs are available. Table 2 provides a feature matrix for common Bluetooth SDKs.

SDK	Vendor	Licensing	.NET	.NET CF	Stacks Supported	Protocols/Profiles Supported
32feet.NET [5]	In The Hand Ltd	Free	Yes	Yes	Microsoft	SDP, OBEX, SPP, DUN, OPP, FTP (Partial), RFCOMM
BlueTools SDK [6]	Franson Technology AB	Licensed	Yes	Yes	Microsoft WIDCOMM	SDP, RFCOMM, OBEX, SPP
BTAccess [7]	High Point Software	Licensed	No	Yes	Microsoft WIDCOMM	SDP, SPP, DUN, PAN, LAP
iAnywhere Blue SDK [8]	Sybase	Licensed	No	Yes	Sybase Protocol Stack	GAP, SDP, SPP, DUN, FAX, OBEX, FTP, Hands-free, Headset, PBAP, PANU, HCRP, BNEP, HID
iAnywhere Blue Manager Suite [8]	Sybase	Licensed	Yes	No	Sybase Protocol Stack	BIP, DUN, FTP, GAP, HCRP, HID, HFP, OPP, PAN, SPP, RFCOMM, OBEX, SDP
BlueSoleil [9]	IVT Corporation	Free	No	No	BlueSoleil	PAN, SPP, OPP, FTP, FAX, A2DP, AVRCP, Headset, Hands-free

Table 2: Feature Matrix for Bluetooth SDKs

4.2.1 Problems with SDK Libraries

Though SDKs ease the burden on the developer, they have inherent problems. The interfaces exposed by SDKs are proprietary and not compatible with each other. This is due to a lack of standard interface definitions. SDKs typically support specific Bluetooth stacks (e.g., Microsoft and WIDCOMM), or only provide features common to all supported stacks. The richness of features available in the individual Bluetooth stacks is thus lost. Since SDKs are not supported by hardware vendors, bugs such as data loss are common when communicating with specific devices. This may be due to internal buffering in the SDKs. There are additional licensing cost associated with SDK's as very few of them of them are free.

Applications should be designed with the flexibility to allow switching SDKs if necessary. A switch may be necessary due to various reasons such as new requirements that are not supported by the SDK that the application is currently using, or lack of support in the SDK for the Bluetooth stack on the target platform. A standard application-level interface for Bluetooth SDKs will allow developers to easily migrate applications across SDKs. The standard could include a core set of commonly used functions to communicate with sensors such as connect, pair, read and write data etc, and allow for extensions to support features unique to individual sensors. A similar approach has been used in the Wireless Extensions for Linux [20] which extends the core networking interface to wireless networking devices. The Wireless Extensions allows the developer to communicate to diverse networking devices in a standard and uniform way.

5. EXPERIMENTAL EVALUATION

5.1 Data Time-stamping

To understand the performance of time-stamping data on the host platform (see Section 3.1.4), we examine the impact of time jitter introduced by the host through experimentations. A PulseOx sensor (Nonin 4100 [18]) generates data at 75 samples/sec and sends a data frame consist of 25 samples every 333 ms to a PC (2GHz dual-core CPU, 2GB RAM, Windows XP). The PC software time-stamps the data frame upon receiving it from the Bluetooth stack. We obtain the timestamp skew by calculating the difference between the expected timestamp of a data frame estimated based on the sampling rate and the receive time of the first data frame, and the timestamp generated by the host software.

Figure 2 presents the probability distribution of the timestamp skews measured on a host with the WIDCOMM stack and Microsoft stack. We collected data for several 15-minute runs performed at different times with the same setup. For space considerations, only data from one run is shown in the figure (each run resulted in a similar distribution). In the plot, negative values indicate the time lag of the timestamps in milliseconds, while positive values indicate that the timestamps are getting ahead due to the time jitters in operations such as buffering and data transmissions. As shown in the figure, a large portion of the timestamping errors, i.e., 63% for MS stack, 90% for WIDCOMM stack, are time lags or delay. The time skews range from -1600ms to 2000ms. The WIDCOMM skew is heavily distributed around -800ms while MS skews are more uniformly distributed around -200ms. It is interesting to note that different Bluetooth stacks impact timestamp performance differently. Overall, in our setup, the MS stack introduces less time jitter than the WIDCOMM stack. As shown in the figure, more than 60% of the timestamp skews are less than 400ms on MS stack while on WIDCOMM more than 80% of the timestamp skews are larger than 400ms.

Note that MS Windows is not a real-time OS, hence a large part of the time jitters can be attributed to the OS scheduling and the interactions between the OS and the Bluetooth stacks. Other factors such as sensor hardware specification, data encoding scheme and transmission rates also impact the time-stamping performance on various levels based on our experiments with different type of streaming devices. The result highlights the need to support accurate time-stamping on the device that generates the data to eliminate the latencies introduced at different layers of the system.

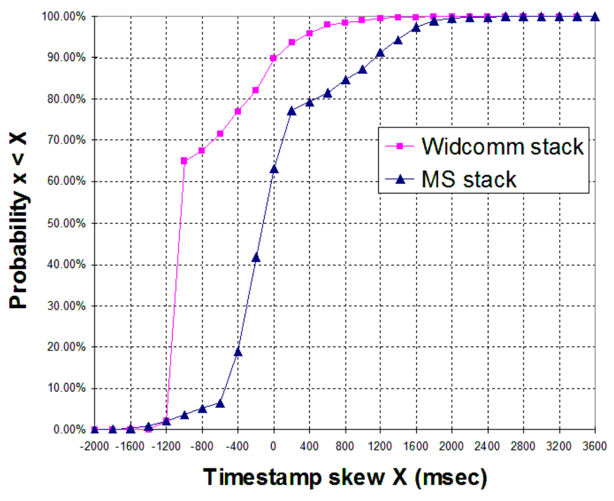

Figure 2: Cumulative Distribution Function (CDF) of timestamp skews

5.2 Reconnection and Link Supervision Timeout

A number of context aware applications interact with on-body sensors. Prior research has shown that on-body sensor communication is highly sensitive and dependent on the relative positions of the sensors and aggregator [19]. Sensors frequently lose connectivity to the aggregator, and the disconnect time periods may vary from under a second to minutes.

During periods of intermittent connectivity, the application may need to disconnect and reconnect to the sensors to restore data streaming. As discussed in Section 3.1.3, the Link Supervision Timeout (LST) defines the length of time the Bluetooth stack monitors a link for data before disconnecting the sensor. Overheads due to disconnect and reconnect latencies can be reduced by choosing optimal values for the LST. Figure 3 plots the overhead as a function of the LST. The overhead is calculated by taking the ratio between the data lost due to the disconnect/reconnect latencies and the net lost during the link vulnerable period. Note that the latencies include processing delay in OS, application and the Bluetooth stack. In our application, the averages disconnect/connect delay was measured to be ~1.5 seconds each. Note that the overhead, as we define it, is independent of the sampling frequency of the sensor. To quickly detect disconnection, a smaller LST is preferable. However, as shown in Figure 3, the overhead for LST in the sub-second range is significantly larger than the rest of the values. Specifically, the overhead levels off at around 3.5 seconds, which marks an optimal LST that balance the tradeoff between overhead and quick disconnection detection. Figure 3 also shows the overhead assuming the MCAP reconnect feature is supported. Assuming the reconnect delay in MCAP is 100 milliseconds, the plot shows that the data loss can be reduced significantly and the LST can be as low as ~500 milliseconds.

6. CONCLUSION

Emerging context aware applications often utilize a constellation of OTS sensors for making sense of the environment and situation of the user. As the diversity and number of sensors increases, system designer and developer are facing a multitude level of challenges in integrating these sensors into their solution. Our contributions in this paper are three-fold:

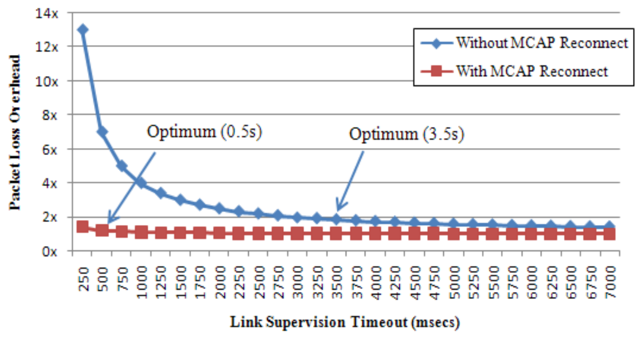

Figure 3: Optimizing Link Supervision Timeout

1) We investigate and report the challenges and solutions of using the existing Bluetooth standard and implementations to support rapid development of classes of context aware applications.
2) We provide useful guidelines and experimental evaluation results to system designers for developing future context aware applications.
3) We also recommend specific modification of Bluetooth standards and implementations, including calls for sensor manufacturers to revise current practices in cases where existing standard features have not been adopted.

7. ACKNOWLEDGMENTS

The authors wish to thank Robert A. Lowe (M.D., M.P.H.), Jim Bryan (M.D), Merlin Curry (EMT-P), Richard Harper (M.D., M.S.), Ritu Sahni (M.D., M.P.H.), Tim Seidel (EMT-P) and Robert Norton (M.D.) for their valuable input in understanding the communication requirements for clinical applications. We also thank Mark Yarvis and the reviewers for their insightful comments that helped to improve the quality of the paper.

8. REFERENCES

[1] Bluetooth SIG, "Bluetooth Core Specification 2.1 + EDR" http://bluetooth.com/NR/rdonlyres/F8E8276A-3898-4EC6-B7DA-E5535258B056/6545/Core_V21__EDR.zip

[2] Bluetooth Medical Devices WG, "Health Device Profile" http://bluetooth.com/NR/rdonlyres/F507EB36-BAB6-4A2F-8121-082D0C2A493A/7759/HDP_SPEC_V10.pdf

[3] Bluetooth Medical Devices WG, "Multi-Channel Adaptation Protocol". http://bluetooth.com/NR/rdonlyres/C04E081E-7E85-4E51-9C4D-3624A07E6715/7760/MCAP_SPEC_V10.pdf

[4] Bluetooth SIG Website. http://www.bluetooth.com/bluetooth/

[5] In The Hand Ltd, 32Feet.NET Bluetooth SDK Website. http://32feet.net/

[6] Franson Technology AB, BlueTools SDK Website. http://franson.com/bluetools/

[7] High Point Software, BTAccess SDK Website. http://www.high-point.com/

[8] Sybase iAnywhere , iAnywhere Blue SDK Website. http://www.sybase.com/products/allproductsa-z/mobiledevicesdks/bluetoothsdks

[9] IVT Corporation, BlueSoleil PC Platform SDK Website. http://www.bluesoleil.com/index.asp

[10] Microsoft Corporation, Microsoft Bluetooth Stack Website. http://msdn.microsoft.com/en-us/library/aa938547.aspx

[11] Broadcom Corporation, WIDCOMM Bluetooth Stack Website. http://www.broadcom.com/products/bluetooth_update.php

[12] Toshiba, Toshiba Bluetooth Stack for Windows XP/Vista Website. http://www.csd.toshiba.com/cgi-bin/tais/support/jsp/askIris.jsp

[13] MindTree Ltd, EtherMind PC Website. http://www.mindtree.com/randdservices/research/ethermind_stack_profile.html

[14] Broadcom Corporation, BTW SDK Website http://www.broadcom.com/products/bluetooth_sdk.php

[15] J. Bray and C.F. Sturman, "Appendix: Bluetooth 1.1 Updates", in Bluetooth – Connect Without Cables, Second Edition, Prentice Hall, 2002.

[16] B. Yener and F. Sivrikaya , "Time Synchronization in Sensor Networks: A Survey", *IEEE Network Magazine*, vol. 18, issue 4, pages 45-50, July/August 2004.

[17] IEEE Std 11073-20601 ™- 2008 Health Informatics - Personal Health De*vice Communication - Application P*rofile - Optimized Exchange Protocol - version 1.0 or later

[18] Nonin Medical – "SpO2 Solutions" http://www.nonin.com/productsList.asp?PageID=2000&sec=1&sub=0

[19] R. Shah, L. Nachman and C-Y. Wan, "On the performance of Bluetooth and IEEE 802.15.4 radios in a body area network", in *Proc of BodyNets 2008*.

[20] Jean Tourrilhes, "Wireless Extensions for Linux", http://www.hpl.hp.com/personal/Jean_Tourrilhes/Linux/Linux.Wireless.Extensions.html

[21] R. W. Woodings, D. D. Joos, T. Clifton and C. D. Knutson. "Rapid Heterogeneous Ad Hoc Connection Establishment: Accelerating Bluetooth Inquiry Using IrDA", in *Proc of Wireless Communications and Networking Conference 2002*.

[22] R. Guerin, E. Kim and S. Sarkar, "Bluetooth Technology Key Challenges and Initial Research", in *Proc of SCS Communication Networks and Distributed Systems Modeling and Simulation CNDS 2002.*

[23] T. Gao, C. Pesto, M. Welsh, etc., "Wireless Medical Sensor Networks in Emergency Response: Implementation and Pilot Results", in *Proc of IEEE International Conference on Technologies for Homeland Security 2008.*

[24] K. Lorincz, D. Malan, T. R. F. Fulford-Jones, A. Nawoj, A. Clavel, V. Shnayder, G. Mainland, S. Moulton, and Matt Welsh, "Sensor Networks for Emergency Response: Challenges and Opportunities", in *IEEE Pervasive Computing, Special Issue on Pervasive Computing for First Response*, Oct-Dec 2004.

[25] M. Gerla, R. Kapoor, M. Kazantzidis and P. Johansson, "Ad hoc Networking with Bluetooth", in *Proc of WMI'01*, Rome, Italy, July. 2001

[26] S. Jung, A. Chang and M. Gerla, "Comparison of Bluetooth Interconnection Methods using BlueProbe", in *Proc of The Second International Workshop On Wireless Network Measurement (WiNMee 2006)*, April 2006.

[27] P. Johansson, M. Kazantzidis, R. Kapoor and M. Gerla, "Bluetooth: An Enabler for Personal Area Networking", in *IEEE Network Magazine*, September. 2001.

[28] L. L. Bello and O. Mirabella, "Clock Synchronization Issues in Bluetooth-based Industrial Measurements", in *Proc of IEEE International Workshop on Factory Communication Systems 2006*, June 2006.

[29] M. Ringwald and K. Romer, "Practical Time Synchronization for Bluetooth Scatternets", in *Proc of Fourth International Conference on Broadband Communications, Networks and Systems 2007*.