Selections from:
Bluetooth Low Energy - The Developer's Handbook
Robin Heydon
Prentice Hall, 2012

Chapter 2. Basic Concepts

In protocol design, perfection has been reached not when there is nothing left to add, but when there is nothing left to take away.
—IETF RFC 1925, Rule 12

To understand Bluetooth low energy is to understand how low power consumption can be achieved in a short-range wireless system. The most basic design decisions are all built around enabling low power consumption in typical use cases.

Bluetooth low energy is not trying to optimize Bluetooth classic; instead, it is targeted at new market segments that haven't previously used open wireless standards. These market segments are those that require devices to send a few octets of data from once a second to once every few days. These are monitoring and control applications that perform tasks, such as detecting whether windows are open or closed for Smart Home heating applications, turning on and off appliances in response to electricity price fluctuations, or changing to a different TV channel.

2.1. Button-Cell Batteries

Button-cell batteries are the primary design goal for Bluetooth low energy. These batteries (see Figure 2–1) have very strict limits on how they can be used. The figure shows a CR2032, although other battery sizes are also available. The "CR" part of the battery label indicates it's a 3-volt battery made using lithium manganese dioxide. The "20" denotes that the battery is 20mm in diameter, and "32" specifies that it's 3.2mm in height.

Figure 2–1. A button-cell battery.

For such small batteries, the maximum energy that can be stored in one manufacturer's battery is very similar to any other. A typical CR2032 will have a

quoted energy capacity of 230mAh at 3 volts. Just to put this into context, this is about the amount of energy required to power a human being for just over 20 seconds. So by the time you've read this paragraph, you'll have used more energy than a typical CR2032 has available to power a Bluetooth low-energy device for a few years.

Even though 230mAh is a comparatively tiny amount of energy, a device will never be able to obtain all of it. First, the energy available is dependent on the temperature of the battery. The colder the battery, the less energy will be available. A button-cell battery at $0^{o}C$ would only be able to provide 80 percent of the energy that is available at room temperature.

Second, if the battery is used aggressively, the total energy available will be significantly reduced. Typically, most button-cell batteries have a peak current that should not be exceeded without damaging the battery. This is typically 15mA. If this high-level current is drawn from the battery for extended periods of time, the total energy available would be reduced. Therefore, any successful radio design would need to manage this and allow the battery time to recover after a large or long current draw.

Finally, the battery itself has its internal leakage current that must also be taken into account. Even if a device is drawing no power from the battery, the battery is still losing charge. When the battery is used sparingly, the leakage current will start to become significant in the total energy budget.

2.2. Time Is Energy

Another basic concept that is used throughout the design of Bluetooth low energy is that time is energy. If the radio is doing something, even if it's nothing more than checking whether it needs to send or receive something, it's using energy. Consequently, it is important to reduce the time required to do anything useful.

A number of important and repetitive actions must be optimized. These include robustly discovering devices, connecting to devices, and sending data. By reducing the time required for these activities, the energy consumption is reduced, lengthening the life of the battery.

Robust device discovery requires a minimum of two devices: a device that is looking for other devices, and one or more devices that are discoverable. In Bluetooth low energy, for a device to be discoverable, it must transmit a very short message three times every few seconds, and if it needs to see if any other device wants to talk to it, it must listen immediately after broadcasting its message. A device that is looking for devices opens up its receiver and listens for devices that are transmitting.

Three transmits are done because three frequencies are used for robustness. The number three is chosen as a compromise between robustness and low power. If the number of frequencies was just one, like a lot of other technologies, then as soon as that frequency is blocked, the whole system would break. If the number of frequencies was, for instance, 16, the device would spend so much time just

transmitting that it would not be low power anymore.

The choice about which device transmits and receives is also very deliberate. To search for a device that is transmitting requires you to listen for a long period of time; this uses a lot of energy and therefore should be done on the device with the bigger energy budget or a good reason to use the low-energy device. In Bluetooth low energy, the discoverable devices transmit, and the devices that are looking for the other devices receive.

The packet itself is very short. Short packets are good for three reasons. First, by using efficient encoding, short packets can send the same quantity of data faster, using less energy. Second, by restricting the devices to only use short packets, the requirements to constantly recalibrate the radio within the controller as the packet is transmitted are removed. Radios, when transmitting or receiving, will heat up, changing the characteristics of the silicon chip, and therefore changing the frequency of the transmissions. If the packets are kept short, the chip doesn't have enough time to heat up; thus, this energy-expensive procedure can be ignored. In addition, the requirement for a short packet also reduces the chip's peak power consumption a little. Finally, you can get more energy out of a button-cell battery by taking it in short-duration bursts and not a long continuous draw of current. Therefore, using several short packets with a sufficient space between them to allow the battery time to recover is better for the battery than using one longer packet.

2.3. Memory Is Expensive

Everybody knows that the more memory a computer has, the more expensive it is. However, every little bit of memory in a computer not only costs money but also costs energy. Memory typically requires dynamic refreshing—every so often the memory in the chip is refreshed. This dynamic refreshing requires energy. So the more memory that a device requires, the more energy is required to power the device. Therefore, the whole of Bluetooth low energy has been designed to reduce the amount of memory that is required in every layer.

For example, keeping the packets small in the Link Layer helps because it reduces the memory requirements for the radio when transmitting and receiving packets. For example, the Attribute Protocol Layer does not require any packets larger than 23 octets to be processed. It also does not require any state information to be saved between transactions. All this reduces the memory needed to do something useful.

Another burden for memory is the multitude of protocols that are required to be active when a device can do multiple things. For example, imagine a headset that does hands-free, remote control, and battery status reporting. If each of these use cases required a separate protocol, the memory required for each of those protocols would have to be added together. In Bluetooth low energy, there is only one protocol. The Attribute Protocol is used for name discovery, service discovery, and for reading and writing information required to implement a given use case. By having only one protocol, the overheads of multiple protocols are significantly

reduced.

2.4. Asymmetric Design

One of the obvious design concepts in Bluetooth low energy, once the architecture has been understood, is the asymmetry that is evident at all layers. This asymmetric design is very important because the device with the smaller energy source is given less to do.

At the Physical Layer, there are two types of radios: transmitters and receivers. A device can have both a transmitter and a receiver. However, a device can implement only a transmitter or only a receiver. If one device only has a transmitter, and the other device only has a receiver, this is an asymmetric network.

This asymmetric design is all based on the fundamental assumption that the most resource-constrained device will be the one to which all others are optimized.

At the Link Layer, devices are divided into advertisers, scanners, slaves, and masters. An *advertiser* is a device that transmits packets; a *scanner* is a device that receives the advertiser's packets. A slave is connected with a master, but even here the asymmetry is evident. A slave cannot initiate any complex procedures, whereas a master has to manage the piconet timing, adaptive frequency hopping set, encryption, and a number of other complex procedures. The slave only does what it is told and doesn't have to perform complex processing at all. This keeps the slave very simple and therefore low cost, low memory, and using the lowest possible power.

At the Attribute Protocol Layer, the two types of devices are called client and server. The server holds data and the client sends requests to the server for this data. The server, like the slave at the Link Layer, just does what it is told. The client has the hard job of working out what data the server has and how to use it.

Even the security architecture for low energy is asymmetric. The security architecture works on a key distribution scheme by which the slave device gives a key to the master device for it to remember. The burden is on the master to remember this bonding information; the slave doesn't have to remember anything. This means that it's simple for a slave device to support security, yet for a master device it is more complex.

This all implies that the most resource-constrained devices will want to be advertisers, slaves, and servers. These types of devices have the lowest possible memory and processing burden; therefore, the asymmetric design is beneficial to the goal of ultra-low power consumption on these types of devices.

The other types of devices—scanners, masters, and clients—have lots of resources to play with. These devices are typically associated with larger batteries, rich user interfaces, and possibly even an electric supply. It is right to move the burden then from the slave to the master, from the advertiser to the scanner, and from the server to the client. This reduces the power consumption of the most resource-constrained devices, to the cost of the most resource-abundant devices.

2.5. Design For Success

So many wireless standards fall down at the first hurdle because a great radio design just doesn't work when it starts to become popular due to congestion from many other radios. If there is one thing that Bluetooth does well, it is operating in a very congested environment. Three times a year, the Bluetooth Special Interest Group (SIG) organizes UnPlugFest testing events at which engineers from many competitive companies come together to test their devices before they are released to the market. These events highlight the fact that Bluetooth still works even when hundreds and even thousands of wireless devices come together in a single hotel ballroom. And Bluetooth low energy has learned from this.

Designing for success means that every person who gets on a crowded commuter train or bus or goes to a busy sports stadium or concert should be able to operate several low-energy devices. This means that thousands of devices could be within a few meters of a device, and device discovery and connections should function as expected. It also means that there should be no inherent limit to the number of devices a given device can talk to at the same time. If a device wants to talk to another device, then it should just be able to do that, not worry that there are only seven possible slaves that can be connected at the same time, which is the limit imposed by classic Bluetooth.

Device density is just one metric that was used during the design of the controller. Another was the security system. Any very popular radio system will become a target for people who want to try to break its security. This becomes even more important when monetary value is involved. So state-of-the-art security and encryption engines must be used.

Beyond security, if a person is going to be carrying around many devices, all of which are resource constrained and therefore advertising continuously, the issue of privacy must also be addressed. In Bluetooth low energy, privacy is dealt with as a major design goal. Every connection that is made uses a different signature that has no relation to any identifying information of these two devices. It is not possible to know who is walking down the street by just listening to the packets being transmitted during connections. Also, when advertising, it is possible to use a *private address*, which is a resolvable address that allows a friendly device to resolve the address if they have the identity resolving key but denies unfriendly devices the ability to resolve or track the address.

Another factor taken into account was that when the radio is used everywhere, even a single bit error can become significant. If you have a sewage outflow valve, for example, protecting your nice public park from being swamped by effluent, you really don't want to have a single bit error that causes that sewage outflow valve to open when you really wanted to make sure it was still closed. To protect against this, all packets have a strong cyclic redundancy check (CRC) value that can protect against all 1, 2, 3, 4, 5, and all odd bit errors. Also, if you want more robustness, you can start encryption. Then a different message authentication code is appended to the data to ensure that the data was sent from the device that you think it was sent

from; no attacker will be able to reply to messages to the sewage outflow valve. And for the really vigilant, at the Attribute Protocol Layer, there is the ability to prepare a write into an attribute and only perform the execute of that write after the value to be written has been returned and verified. This means that for this single bit state for the valve, a total of 14 octets of CRC and authentication codes protect this data. Bluetooth low energy is robust.

2.6. Everything Has State

One of the basic concepts behind Bluetooth low energy is that everything has state. This state is exposed by using the Attribute Protocol in an attribute server. The state could be anything: the current temperature, the state of the battery of the device, the name of the device, or the description of where the temperature is being measured.

State doesn't just have to be readable state; it can also be written. A thermostat can have a set-point temperature by which another device can set the temperature to which this room should be heated or cooled. If you can expose state, you can also expose the state of a state machine. By using explicit state machine attributes, the state of the device can be clearly exposed. This offers the capability for clients to disconnect whenever they want, because when they reconnect, they can quickly determine the current state by just reading it.

Some state is variable and can change frequently. To enable an efficient transfer of state information from the server to the client, direct notifications of the state information from the server to the client are possible. These notifications don't require the client to poll the server, allowing very efficient application designs. The battery state could be notified only when something interesting happens; thus, a device wouldn't need to worry about the battery state at all until the notification arrives.

This simple-state–based model makes it possible for a very efficient client server architecture to be constructed. This also allows an object-oriented approach to state to be designed into applications, with reusable data types and service behavior. This reduces the quantity of code that a device needs to contain, which consequently reduces the power consumption of the device because memory doesn't need to be provided for that code. And there is another significant benefit to having less code: fewer bugs. Simpler systems are cheaper and faster to develop. A simpler system also typically contains fewer errors, making it more robust. Finally, simpler systems are easier to maintain. As Robert Browning said, "Less is more."

2.7. Client-Server Architecture

The client-server architecture has one additional basic design element that is fundamental to the design of Bluetooth low energy. When low energy was being designed, the problem of connecting devices to the Internet was considered. It could have been possible to put an Internet Protocol (IP) stack on every single resource-constrained device and just expose all the devices over the Internet. Unfortunately,

even the simplest of IP stacks takes more memory and energy than is desirable on the simplest of devices. Therefore, the decision was made to not allow any IP packets to be routed directly to slave devices.

Instead, smart gateways allow the interconnection between the Internet and very efficient low-energy slaves. This interconnection is possible because of the pure client-server architecture. A server is just a repository of data, and it does not care who the client is. A client could be directly connected to the server or it could be connected via an Internet gateway from the other side of the planet.

This affords the ability for individuals to monitor and control their home when they are on vacation. And given that low energy will be used for everything from security alarms to set-top boxes and heating systems, it would be possible to check that all the windows are secure on the way to the beach, set up a recording of your favorite television program while you're lying back in the sand, and then turn the heat back up while flying home.

The ability to connect with gateways also allows sports and fitness devices to instantly update their associated web sites with their collected data, even before the exerciser has had a chance to finish her drink of water. It would also provide the ability to monitor elderly people so that they can stay in their own homes, safe in the knowledge that people are available should they need help.

The client-server gateway model also enables full Internet security to be used from the client to the gateway and allows the gateway to perform access control, firewall, and authorization of the client before granting it access to anything beyond the gateway. These are proven technologies used today in many homes and businesses.

2.8. Modular Architecture

One basic concept that is often overlooked is future-proofing the architecture. Most wireless standards are created in a rush, trying to get the technology out as quickly as possible, without much concern about how the technology will function in 10 or 20 years' time. This causes problems, because poor architectural decisions made under the duress of "time to market" damage the long-term viability of the platform. To solve this, the Bluetooth SIG created a special architecture working group just for the Generic Attribute Profile–based architecture to ensure a future-proof design.

The main outcome of this group has been the modular service architecture that builds on top of the generic attribute profile. This allows atomic encapsulatable bits of behavior to be wrapped up in a single service and exposed on a device in a standard way. (In this context, an atomic service is one that just does a single thing, and encapsulatable means that it can be separated from other functions and wrapped up by itself.) These services can reference other services, so a battery service can reference a temperature service if the battery has its own temperature sensor; this same temperature service can be reused for a home thermometer, a freezer temperature sensor, or a car engine coolant temperature sensor.

An interesting side effect of this architecture is that the services exposed on a device do not have to be directly related to a given profile. Profiles will require a given set of services on a given device, but that is about as much of a link as needed. This means that if another profile can be created to use a different combination of services on a device, it can combine the existing services in a different way without a problem. This is true, even if that profile was written after the services were designed and implemented in a device.

This is a highly flexible and modular architecture that can enable the building up of ecosystems over time. For example, smart meters could be deployed into homes to allow current and future price information and usage information to be exposed. Later, smart appliances can be deployed that allow themselves to be remotely turned on and off; using the gateway model, this could be controlled outside the home. Even later, a smart energy broker can be deployed that uses the information from the smart meter and the information from the smart appliances to save the homeowners money by scheduling energy use that takes into account the pricing information from the meter.

2.9. One Billion Is a Small Number

Any new technology faces a serious challenge trying to obtain market traction. For a technology to be successful, it has to be low cost. To be low cost, you need volume. To have volume, you need to be successful. Today, the single largest consumer electronics device that is sold is the cell phone. Any technology that makes it into the cell phone will be successful. Bluetooth is the classic example of this. Bluetooth low energy builds on Bluetooth's attach rate in cell phones to create an instant market.

The technology has the opportunity to have over one billion devices in the field within the first couple of years as the cell phone manufacturers update their platforms to include Bluetooth low energy. The interesting thing about this is that it creates a huge market for accessories for phones. And it is not just phones that can have Bluetooth low energy designed in quickly; computers, televisions, in fact, any devices that have Bluetooth classic, are likely to be updated to add Bluetooth low energy because of the extremely low cost associated with incorporating the new technology to an existing Bluetooth system.

2.10. Connectionless Model

Bluetooth classic was all about cable replacement: headset cables, mouse cables, file transfer cables. This implies an architecture where the cost of setting up a link is not that important because the link will be maintained for a few minutes, hours, or even days. The odd second delay at the start of the connection is not that important. Bluetooth low energy changes all this.

The basic concept with low energy is that connections are transient. When you need to do something or check something, you quickly create a connection, do what needs to be done, and then disconnect. A device that is only notifying some state

information once every five minutes would only need the radio on for less than one second a day. This means that the radio is off 99.999 percent of the time; or to four significant digits, the radio is off 100 percent of the time. Any delay in each connection setup will cause a significant increase in power consumption.

Bluetooth low energy can create a connection, send data, and gracefully disconnect in about three milliseconds. This means that many devices that have some state information, but until now couldn't afford to add wireless technology because of the cost of energy requirements, can finally consider adding Bluetooth low energy. Even something as simple as a button can be enabled, possibly using scavenged power, and therefore never need a battery.

2.11. Paradigms

Most successful technology is built around sets of paradigms, and Bluetooth low energy is no different. Bluetooth low energy uses two main architectural paradigms: client-server architecture and service-oriented architecture.

2.11.1. Client-Server Architecture

The client-server architecture is a paradigm by which clients can send requests to servers over a network and the servers send back responses. It is the main paradigm behind the Internet, which is arguably the most successful networking technology ever released.

For example, when you type a URL address into a web browser, it first sends the address to a DNS server. This server responds with the IP address of the server that has been assigned to that name. The client then sends a hypertext transport protocol (HTTP) request to that server and, once connected, sends a request for the server to get the resource identified in the request. The server then responds with the appropriate resource, typically a text file that contains markup (HTML) information about how to display the information.

This file can also include additional URLs with which the client can fetch other resources, such as pictures or other pages. These additional links are really the reason HTML pages are thought of as being linked together into a *web*, hence the terms *web page* and *web server*.

There is a clear distinction between what a server does and what a client does. The server has information, typically in a structured form. This data is really why the server exists. This data can be anything, the current weather in Kona, Hawaii, the time of the next train from Seoul to the airport, or just some inane chatter between friends. The client, on the other hand, doesn't have any data. It just sends requests to servers. Once it receives the replies from a server, it can carry out the task it was assigned to do, such as display information to the user or notify the user that somebody they know has posted something on a wall or tweeted.

The main benefit of the client-server architecture is this defined split between the client and the server. This split is necessary when the different parts of the system are on different devices. By defining one of these parts as a server and one as a

client, the explicit relationship between these two parts of the system can be determined.

The main benefit of this architecture is that it can scale. A client doesn't need to know anything except the URL to be able to access a resource. There can be many clients. Some sites on the Internet will have millions of requests made each day from millions of clients. The server doesn't really care what or where these clients are; it just responds to each request as it comes in.

This server architecture can also be scaled. A single machine responding to millions of requests a day might become overloaded and start to fail. The solution is to place many identical servers, all having access to the same information. This is further assisted when clients are given multiple IP addresses for a single name so that the load is spread evenly among each server. This is known as *load balancing*.

2.11.2. Service-Oriented Architecture

A further abstraction on top of the client-server architecture is the service-oriented paradigm. This is a model that organizes the information in a server into services. These services can be discovered, interacted with, and used with known semantics. This means that the services have a defined behavior that will always produce the same result, given the same preconditions.

This paradigm is the foundation of the most highly successful Internet systems, such as SOAP, REST, COBRA, RPC, Web Services, and so on.

A way to illustrate this is to relate it to a real-world example. Suppose that you have a package that needs to be delivered to another company quickly. The first thing you will probably do is call a courier company, arrange a pickup of the package, and then pay for the service. The key concept is that you know what is going to happen. There is an implicit set of behaviors that the courier company will be following. On any day, given the same package to be delivered, the courier company will do exactly the same thing—deliver it to its destination, on time. This service has known semantics with defined behavior that produces predictable results.

An interesting part of this is that you are interacting with two different people at the courier company: the person who answered the phone and took your request and the delivery driver who collected your package. You also used, although unknowingly, a third person who dealt with the financial transaction. Each of these people provide a subservice that, when combined, results in the primary service offering of the courier company.

Some of these subservices are also generic; they could be used interchangeably by many different types of companies. The processing of financial transactions is something that is done pretty much the same way in every company. Similarly, the function of taking phone calls for picking something up at one location and dropping it off at another could also be applied to taxi companies.

For all this to work, everything must adhere to a set of rules and conventions that are outlined in the sections that follow.

Chapter 3. Architecture

> *That's been one of my mantras—focus and simplicity. Simple can be harder than complex: You have to work hard to get your thinking clean to make it simple. But it's worth it in the end because once you get there, you can move mountains.*
>
> *—Steve Jobs*

The architecture for Bluetooth low energy is fundamentally very simple. As shown in Figure 3–1, it is split into three basic parts: controller, host, and applications. The controller is typically a physical device that can transmit and receive radio signals and understand how these signals can be interpreted as packets with information within them. The host is typically a software stack that manages how two or more devices communicate with one another and how several different services can be provided at the same time over the radios. The applications use the software stack, and therefore the controller, to enable a use case.

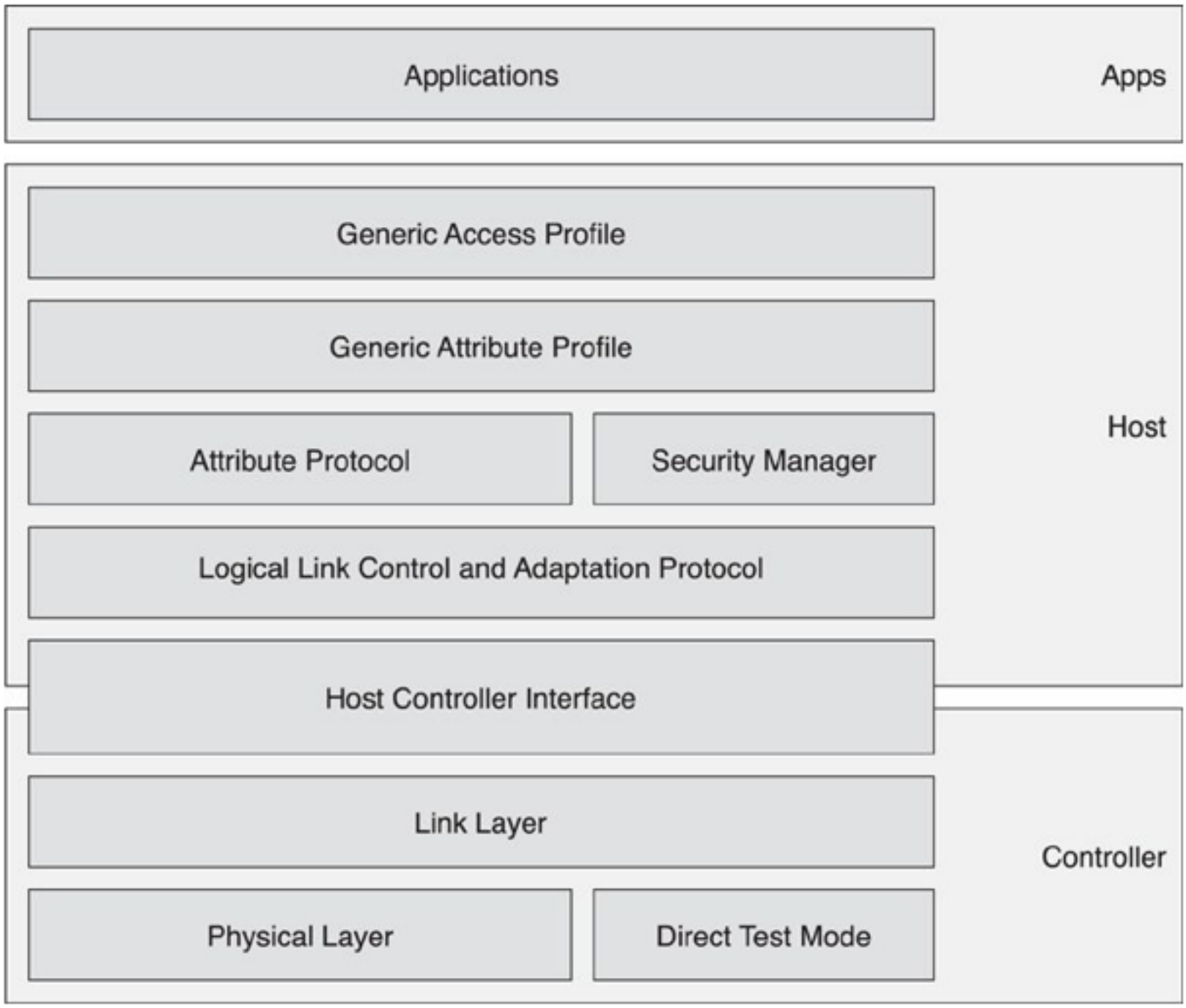

Figure 3–1. The Bluetooth Architecture

Within the controller, there is both the Physical Layer and Link Layer as well as a Direct Test Mode and the lower layer of the Host Controller Interface. Within the host are three protocols: Logical Link Control and Adaptation Protocol, Attribute Protocol, and the Security Manager Protocol. Also within the host are the Generic Attribute Profile, the Generic Access Profile, and modes.

3.1. Controller

The controller is the bit that most people can identify as the Bluetooth chip or radio. Calling the controller a radio, however, is very simplistic; the controller is composed of both analog and digital parts of the radio frequency components as well as hardware to support the transmission and reception of packets. The controller interfaces with the outside world through an antenna and to the host through the Host Controller Interface.

3.1.1. Physical Layer

The Physical Layer is the bit that does the hard work of transmitting and receiving bits using the 2.4GHz radio. To lots of people, the Physical Layer is magical. Fundamentally, it is not magic, but is the simple transmission and reception of electromagnetic radiation. Typically, radio waves can carry information by varying the amplitude, frequency, or phase of the wave within a given frequency band. In Bluetooth low energy, the frequency of the radio waves are varied to allow either a zero or a one to be exposed, using a modulation scheme called Gaussian Frequency Shift Keying (GFSK).

The Frequency Shift Keying part means that ones and zeros are coded onto the radio by slightly shifting the frequency up and down. If the frequency is shifted abruptly to one side or the other at the moment the frequency changes, there is a pulse of energy that spreads out over a wider range of frequencies. So a filter is used to stop the energy spreading too far into higher or lower frequencies. In the case of GFSK, the filter used is shaped like a Gaussian curve. The filter used for Bluetooth low energy is not as tight as the filter used for Bluetooth classic. This means that the low energy radio signal spreads out a little more than the classic radio signal.

This slight widening of the radio signal is useful because it means the radio comes under spread-spectrum radio regulations, whereas the Bluetooth classic radio is governed by frequency-hopping radio regulations. Spread-spectrum radio regulations allow a radio to transmit on fewer frequencies than frequency-hopping radio regulations. Without the more relaxed filter shape, the Bluetooth low energy radio would not be allowed to advertise on just three channels; it would have to use many more channels, which would make the system higher power (as discussed earlier).

This slight widening of the radio signal is referred to as the modulation index. Modulation index describes how wide the upper and lower frequencies used are around the center frequency of a channel. When the radio signal is transmitted, a

positive frequency deviation of more than 185kHz from the center frequency represents a bit with the value 1; a negative frequency deviation of more than 185kHz represents a bit with the value 0.

For the Physical Layer to work, especially when lots of other radios are in the same area transmitting at the same time, the 2.4GHz band is split up into 40 separate RF channels, each 2MHz apart from one another. The Physical Layer transmits information at one bit of application data every one microsecond. For example, to send the 80 bits of data for the string "low energy" formatted in UTF-8 would take just 80µs, although this does not take into account any packet overhead.

3.1.2. Direct Test Mode

Direct Test Mode is a novel approach to the testing of the Physical Layer. In most wireless standards, there is no standard way to get a device to perform standard Physical Layer tests. This leads to the problem of many different companies building their own proprietary methods to test only their Physical Layers. This increases the costs for the whole industry and increases the barriers for an end-product manufacturer to change from one silicon supplier to another quickly.

Direct Test Mode allows a tester to command a controller's Physical Layer to either transmit a sequence of test packets or receive a sequence of test packets. The tester can then analyze the packets received, or the number of packets that the device under test received, to determine if the Physical Layer is working according to the specification. The tester can also measure various RF parameters from received packets to determine if the Physical Layer is compliant with the RF specs. The Direct Test Mode is not just applicable to qualification testing; it can also be used for production line testing and calibration of radios. For example, by quickly commanding a Physical Layer to transmit on a given radio frequency, and measuring the actual transmitted signal, the radio can be tuned to match what it should be doing. This sort of calibration is typically done on every single unit, so having test equipment that can do this efficiently can save product manufacturers money.

3.1.3. Link Layer

The Link Layer is probably the single most complex part of the Bluetooth low energy architecture. It is responsible for advertising, scanning, and creating and maintaining connections. It is also responsible for ensuring that packets are structured in just the right way, with the correctly calculated check values and encryption sequences. To do this, three basic concepts are defined: channels, packets, and procedures.

There are two types of Link Layer channels: advertising channels and data channels. Advertising channels are used by devices that are not in a connection sending data. There are three advertising channels—again, this is a compromise between low power and robustness. Devices use these channels to broadcast data, advertise that they are connectable and discoverable, and to scan and initiate connections. The data channels are only used once a connection has been established and data needs to flow. There are 37 data channels, and they are used through an

adaptive frequency-hopping engine to ensure robustness. The data channels allow data from one device to another to be sent, acknowledged, and, if necessary, retransmitted. Data channels can be encrypted and authenticated on a per-packet basis.

To send data on any of these channels (data or advertising), small packets are defined. A packet encapsulates a small amount of data that is sent from a transmitter to a receiver over a very short period of time. Packets include information to identify the intended receiver, as well as a checksum that ensures that the packet is valid. The basic packet structure is identical between the advertising channels and data channels, with a minimum of 80 bits of addressing, header, and check information included in each and every packet. Figure 3–2 presents an overview of the Link Layer packet structure.

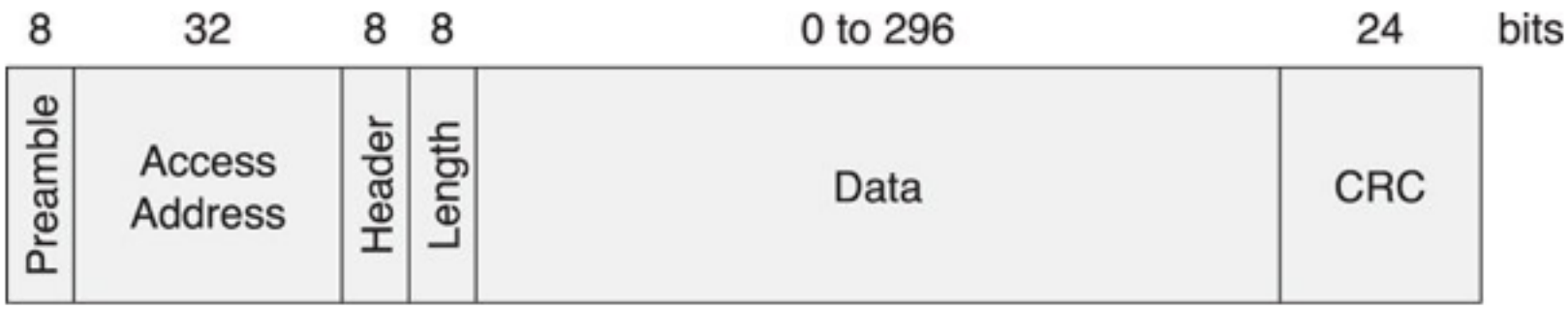

Figure 3–2. The Link Layer packet structure

The packets are optimized to increase their robustness by using an 8-bit preamble that is sufficiently large to allow the receiver to synchronize bit timing and set the radio's automatic gain control; a 32-bit access address that is fixed for advertising packets but completely random and private for data packets; an 8-bit header to describe the contents of the packet; an 8-bit length field to describe the payload length, although not all these bits are used for length because no packet with more than 37 octets of payload is allowed to be sent; a variable-length payload that contains useful data from the application or the host device stack; and finally, a 24-bit cyclic redundancy check (CRC) value to ensure that there are no bit errors in the received packet.

The shortest packet that can be sent is an empty data packet that is 80µs in length, whereas the longest packet is a fully loaded advertising packet that is 376µs in length. Most advertising packets are just 128µs in length, and most data packets are 144µs in length.

3.1.4. The Host/Controller Interface

For many devices, a Host/Controller Interface (HCI) will be provided that allows a host to communicate with the controller though a standardized interface. This architectural split has proven to be extremely popular in Bluetooth classic, for which over 60 percent of all Bluetooth controllers are used through the HCI interface. It allows a host to send commands and data to the controller and the controller to send events and data to the host. It is really composed of two separate parts: the logical interface and the physical interface.

The logical interface defines the commands and events and their associated behavior. The logical interface can be delivered over any of the physical transports, or it can be delivered via a local application programming interface (API) on the controller, allowing an embedded host stack to be included within the controller.

The physical interface defines how the commands, events, and data are transported over different connection technologies. The physical interfaces that are defined include USB,[1] SDIO,[2] and two variants of the UART.[3] For most controllers, they will support just one or possibly two interfaces. It should also be considered that to implement a USB interface requires lots of hardware, and the interface is not the lowest power interface, so it would not typically be provided on a Bluetooth low energy single-mode controller.

Because the host controller interface has to exist on both the controller and the host, the part that is in the controller is typically called the lower-host controller interface; the part that is in the host is typically called the upper-host controller interface.

3.2. The Host

The host is the unsung hero of the Bluetooth world. The host contains multiplexing layers, protocols, and procedures for doing lots of useful and interesting things. The host is built on top of the upper-host controller interface. On top of this is the Logical Link Control and Adaptation Protocol, a multiplexing layer. On top of this are two fundamental building blocks for the system; the Security Manager that does everything from authentication and setting up secure connections and the Attribute Protocol that exposes the state data on a device. Built on the Attribute Protocol is the Generic Attribute Profile that defines how the Attribute Protocol is used to enable reusable services that expose the standard characteristics of a device. Finally, the Generic Access Profile defines how devices find and connect with one another in an interoperable manner.

There is no defined upper interface for the host. Each operating system or environment will have a different way of exposing the host APIs, whether that be through a functional or object-oriented interface.

3.2.1. Logical Link Control and Adaptation Protocol

The Logical Link Control and Adaptation Protocol (also referred to as L2CAP) is the multiplexing layer for Bluetooth low energy. This layer defines two basic concepts: the L2CAP channel and the L2CAP signaling commands. An L2CAP channel is a single bidirectional data channel that is terminated at a particular protocol or profile on the peer device. Each channel is independent and can have its own flow control and other configuration information associated with it. Bluetooth classic uses most of the features of L2CAP, including dynamic channel identifiers, protocol service multiplexers, enhanced retransmission, and streaming modes. Bluetooth low energy just takes the absolute minimum of L2CAP.

In Bluetooth low energy, only fixed channels are used: one for the signaling

channel, one for the Security Manager, and one for the Attribute Protocol. There is only one frame format, the B-frame; this has a two-octet length field and a two-octet channel identifier field, as illustrated in Figure 3–3. This is the same frame format that classic L2CAP uses for every channel until the frame formats are negotiated to something more complex. For example, in Bluetooth classic, it is possible to have frame formats that include additional frame sequencing and checks. These are not needed in Bluetooth low energy because the checks at the Link Layer are strong enough to not need additional checks, and the simple Attribute Protocol has no need for out-of-order delivering of packets from multiple channels. By keeping the protocols simple and doing sufficient checks, only one frame format was required.

Figure 3–3. The L2CAP packet structure

3.2.2. The Security Manager Protocol

The Security Manager defines a simple protocol for pairing and key distribution. Pairing is the process of attempting to trust another device, typically by authenticating the other device. Pairing is typically followed by the link being encrypted and the key distribution. Using key distribution, shared secrets can be distributed from a slave to a master so that when these two devices reconnect at a later date, they can quickly prove their authenticity by encrypting using the previously distributed shared secrets. The Security Manager also provides a security toolbox for generating hashes of data, generating confirmation values, and generating short-term keys used during pairing.

3.2.3. The Attribute Protocol

The Attribute Protocol defines a set of rules for accessing data on a peer device. The data is stored on an attribute server in “attributes” that an attribute client can read and write. The client sends requests to the server and the server responds with response messages. The client can use these requests to find all the attributes on a server and then read and write these attributes. The Attribute Protocol defines six types of messages: 1) requests sent from the client to the server; 2) responses sent from the server to the client in reply to a request; 3) commands sent from the client to the server that have no response; 4) notifications sent from the server to the client that have no confirmation; 5) indications sent from the server to the client; and 6) confirmations sent from the client to the server in reply to an indication. So, both client and server can initiate communication with messages that require a response, or with messages that do not require a response.

Attributes are addressed, labeled bits of data. Each attribute has a unique handle that identifies that attribute, a type that identifies the data stored in the attribute, and a value. For example, an attribute with the type Temperature that has the value 20.5°C could be contained within an attribute with the handle 0x01CE. The Attribute Protocol doesn't define any attribute types, although it does define that some attributes can be grouped, and their group semantics can be discovered via the Attribute Protocol.

The Attribute Protocol also defines that some attributes have permissions: permissions to allow a client to read or write an attribute's value, or only allow access to the value of the attribute if the client has authenticated itself or has been authorized by the server. It is not possible to discover explicitly an attribute's permissions; that can only be done implicitly by sending a request and receiving an error in response, stating why the request cannot be completed.

The Attribute Protocol itself is mostly stateless. Each individual transaction—for example, a single read request and read response—does not cause state to be saved on the server. This means that the protocol itself requires very little memory. There is one exception to this: the prepare and execute write requests. These store a set of values that are to be written in the server and then executed all in sequence, in a single transaction.

3.2.4. The Generic Attribute Profile

The Generic Attribute Profile sits above the Attribute Protocol. It defines the types of attributes and how they are used. It introduces a number of concepts, including "characteristics," "services," "include" relationships between services, and characteristic "descriptors." It also defines a number of procedures that can be used to discover the services, characteristics, and relationships between services, as well as read and write characteristic values.

A service is an immutable encapsulation of some atomic behavior of a device. This is a long stream of very complex words, but it is a very simple concept to understand. Immutable means that once a service is published, it cannot change. This is necessary because for a service to be reused it can never be changed. As soon as a service's behavior changes, version numbers and other awkward setup procedures and configuration take time and therefore become the antithesis of a connectionless model, one of the basic concepts behind Bluetooth low energy.

Encapsulation means expressing features of something succinctly. Everything about a given service is enclosed and expressed through a set of attributes in an attribute server. Once you know the bounds of a service on an attribute server, you know what information that service is encapsulating. Atomic means of or forming a single irreducible unit or component of a larger system. Atomic services are important because the smaller the server, the more likely it is to be reusable in another context. If we created complex services that had multiple, possibly related behaviors, the chance of these being reused is significantly reduced.

Behavior means the way something acts in response to a particular situation or

stimulus. For services, the behavior means what happens when you read or write an attribute, or what causes the attribute to be notified to the client. Explicitly defined behavior is very important for interoperability. If a service is specified with poorly defined behavior, each client might act in a different way when interacting with the service. The services might then act differently depending on which client is connecting, or more important, the same service on different devices will act differently. As soon as this becomes entrenched in the devices, interoperability is destroyed. Therefore, explicitly defined behavior that is testable, even for erroneous interactions, promotes interoperability.

Service relationships are key to the complex behaviors that devices expose. A service is atomic by nature. Complex behaviors should not be exposed in just a single service. Take, for example, a device that can measure the room temperature by exposing a temperature service. The device might be powered by a battery so it would expose a battery service. However, if the battery also has a temperature sensor, we should be able to expose another instance of the temperature service on the device. This second temperature service needs to be related to the battery so that a client can determine that relationship. This is shown in Figure 3–4.

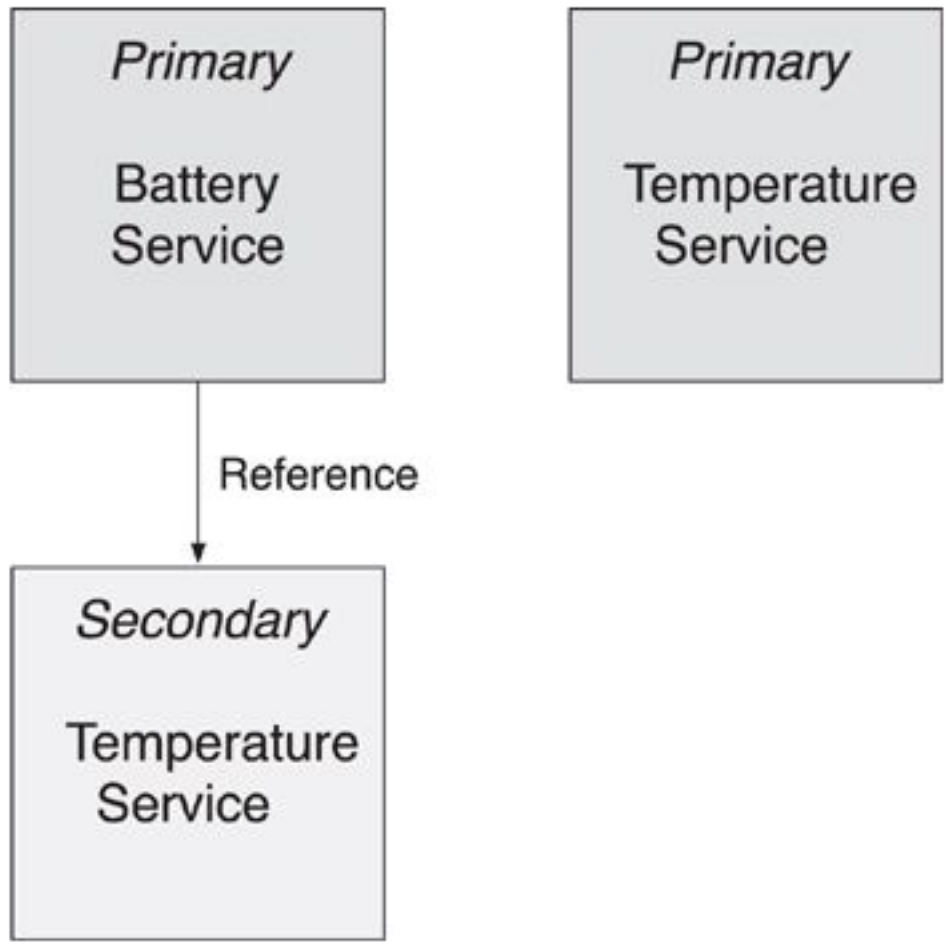

Figure 3–4. Complex service relationships

To accomodate complex behaviors and relationships between services, services come in two types: primary services and secondary services. The type of a service is not typically dependent on the service itself but on how that service is used in a device. A primary service is one that exposes what the device does, from the perspective of the user. A secondary service is one that is used by a primary service or another secondary service to enable it to provide its complete behavior. In the previous example, the first temperature service would be a primary service, the

battery service would also be a primary service, whereas the second instance of the temperature service—the temperature of the battery—would be a secondary service referenced from the battery service.

3.2.5. The Generic Access Profile

The Generic Access Profile defines how devices discover, connect, and present useful information to the users. It also defines how devices can create a permanent relationship, called *bonding*. To enable this, the profile defines how devices can be discoverable, connectable, and bondable. It also describes how devices can use procedures to discover other devices, connect to other devices, read their device name, and bond with them.

This layer also introduces the concept of privacy by using resolvable private addresses. Privacy is important for devices that are constantly advertising their presence so that other devices can discover and connect to them. Devices that want to be private, however, must broadcast by using a constantly changing random address so that other devices cannot determine which device it is by listening, or which device is moving around by tracking its current random address over time. However, to allow devices that are trusted to determine if it is nearby, and to allow connections, the private address must be resolvable. The Generic Access Profile, therefore, defines not only how private addresses are resolvable but also how to connect to devices that are private.

3.3. The Application Layer

Above the controller and the host is the Application Layer. The Application Layer defines three types of specifications: characteristic, service, and profile. Each of these specifications is built on top of the Generic Attribute Profile. The Generic Attribute Profile defines grouping attributes for characteristics and services, and the applications define the specifications that use these attribute groups.

3.3.1. Characteristics

A characteristic is a bit of data that has a known format labeled with a Universally Unique Identifier[4] (UUID). Characteristics are designed to be reusable, and therefore have no behavior. As soon as behavior is added to something, it limits its reuse. The most interesting thing about characteristic specifications is that they are defined in a computer-readable format rather than as human-readable text. This gives computers the ability, when they see a characteristic used for the first time, to download this computer-readable specification and use it to display these characteristics to the user.

3.3.2. Services

A service is a human-readable specification of a set of characteristics and their associated behavior. The service only defines the behavior of these characteristics on a server; the service does not define the client behavior. For many services, the client behavior can be implicitly determined by the service's server behavior.

However, for some services, there might need to be more complex behavior in the client that must be defined. This client behavior is defined in profiles, not in the services.

Services can include other services. The parent service can only define the services that are included; it cannot change the characteristics in these included services or change the behavior of these services. The including service, however, can describe how multiple included services interact with each other.

Services come in two variants: primary and secondary, as noted in Section 3.2.4. The primary or secondary nature of a service can be defined in a service specification or can be left up to the profile or an implementation. Primary services are those that embody what a given device does—it is these services that the user would understand that the device does. Secondary services are those that assist the primary services or other secondary services.

Services do not describe how devices connect to each other to find and use services. Services only describe what happens when a characteristic is read or written, or when it is notified or indicated. Services do not describe what Generic Attribute Profile procedures are used to find the service, the characteristics within a service, or how the characteristics are used by a client.

3.3.3. Profiles

Profiles are the ultimate embodiment of a use case or application. Profiles are specifications that describe two or more devices, with one or more services on each device. Profiles also describe how the devices should be discoverable and connectable, thereby defining what topology is necessary on each device. Profiles also describe the client behavior for finding the service, finding the characteristics of the service, and using the service to enable the functionality required by the use case or application.

There is a many-to-many mapping of profiles to services, as illustrated in Figure 3–5. A service can be used by many profiles to enable a given behavior on a device. The behavior of a service is independent of which profile is using this service at this time. Application stores can be given the list of services that a device supports and find the set of applications from the store that use these services. This flexibility enables a plug-and-play model that has worked so well for the universal serial bus.

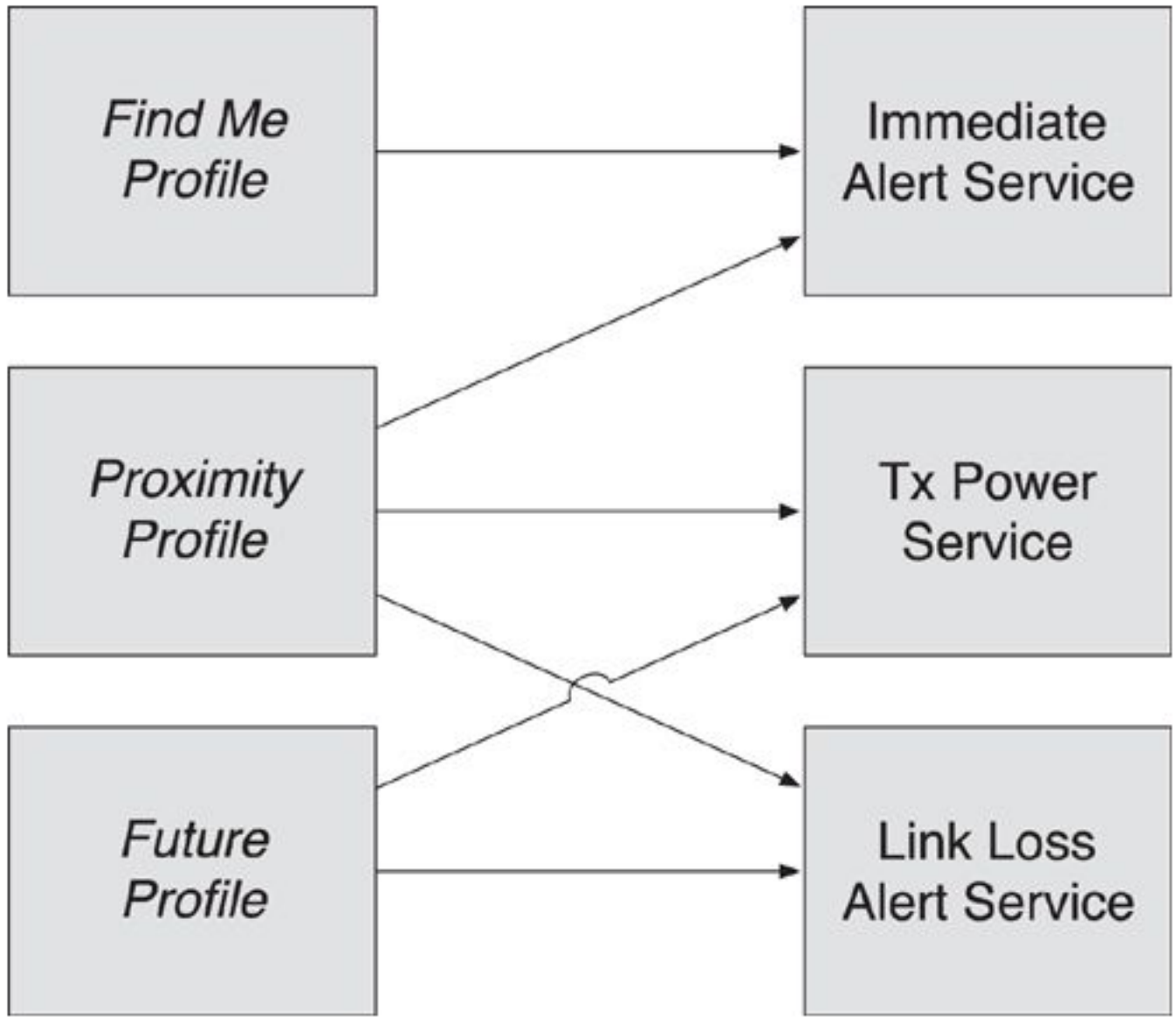

Figure 3–5. Complex profile service relationships

3.4. Stack Splits

It is possible to build a Bluetooth low energy product by using multiple different stack splits. The specification defines one stack split using the host controller interface between the controller and the host, but you can use many other different stack splits.

3.4.1. Single-Chip Solutions

The simplest stack split that is possible with Bluetooth low energy is the single-chip solution, as shown in Figure 3–6. This has no stack splits; all parts of the product are packed into a single chip. This chip includes the controller, the host software, and the applications. This is the ultimate in low-cost products, only requiring a source of power, an antenna, some hardware to interface to—for instance, buttons and lights—and some additional discrete components.

Chapter 4. New Usage Models

All of the books in the world contain no more information than is broadcast as video in a single large American city in a single year. Not all bits have equal value.

—Carl Sagan

Bluetooth low energy enables a new way of using wireless technology. The main new models are based around the advertising model and include presence detection, broadcasting of data, and connectionless models. They also include gateways from devices to the Internet.

4.1. Presence Detection

The most interesting new wireless model enabled by Bluetooth low energy is presence. Presence means the state or fact of existing, occurring, or being present in a place or thing. Using the advertising model, devices can passively scan in the background for other devices that are broadcasting. The devices that are advertising might be just advertising their address, or they could be advertising some presence-based data.

Advertising is a new mode of operation defined in the Link Layer. With it, devices can periodically transmit their identity and a small amount of information. This model is possible because the modulation index of the radio has been increased and the 2.4GHz band regulations permit wider radio signals to be sent using a non-frequency-hopping radio. Because the radio doesn't employ frequency hopping, fewer channels are needed for devices to be connectable and discoverable. This means that it is much more efficient to advertise and scan.

Scanning is possible in two modes: active and passive. Active scanning requires the scanner to request more information from advertisers, to obtain additional static data. Passive scanning just requires the scanner to listen for advertising packets. Once an advertising packet is received by the Link Layer, it can be sent to the host.

The host can use the information about what devices are nearby to determine where it is. For example, if the host discovers a car, then the host can determine that it is in or near a car and change its behavior accordingly, perhaps by connecting to the car. Similar use cases around the home, in the office, or at a café are also possible. It is this automatic determination in the background using passive scanning that allows a device to automatically change its behavior based on where it is.

Presence, as just described, is about a mobile device determining where it is. Another type of presence is about static devices being able to determine what devices are in a given location. Probably the most useful benefit of this would be to find somebody or something in a large office building, for example. The devices or

people that want to be tracked would advertise infrequently, and devices in each room would monitor which devices they can detect. This information can then be communicated to a central device to determine location. You can use this to automatically route phone calls to the nearest phone or to track employees during an emergency evacuation of a building.

4.2. Broadcasting Data

The advertising model also allows a small quantity of data to be broadcast—a very small amount, just a few tens of octets of data—but the ability to broadcast this small bit of information to any device that is listening in the area is incredibly valuable. As stated earlier, the ability to determine where a device is, based on what devices are broadcasting, is a useful function in its own right. However, this relies on having some way to map the device that is broadcasting to a physical location. Data broadcasting helps with this mapping.

You can use broadcasting to transmit many different types of useful data. There are three main areas for which broadcasting data can help the user experience: initial connection setup, advertising, and broadcasting information.

To help with an initial connection setup, devices can broadcast data about what type of device they are and that they want to connect to a device with a complementary set of services or profiles. For example, when you remove a television from its box and switch it on for the first time, it starts to search for a remote control. When you install the batteries in the remote control, it starts to advertise that it is looking for the television.

The television receives this broadcast data, connects to the remote control, automatically pairs with it, and then allows the remote control to talk to it securely. This means that, from the consumer's perspective, they turn on the television for the first time, put the batteries in the remote control, and then press buttons on the remote to control the television; no connect buttons and no pairing menus.

Advertising is a useful tool for many organizations. With it, consumers have the ability to discover real world-services from over 100 meters away. An obvious place where advertising using a free wireless technology is useful is at international airports and railway stations. The ability to advertise gate or track details for flights or trains gives travellers who don't want to spend lots of money on roaming charges or Wi-Fi Internet access an alternative way to gather information. Simple bus stops can also advertise when the next bus will arrive and where the bus is heading.

It is also possible to broadcast information that is gathered locally by a device. For example, a temperature sensor could broadcast the temperature to any device that is currently listening for temperature information. This is most useful when information is being sensed that changes rapidly and the information is useful for multiple devices.

4.3. Connectionless Model

One of the biggest changes from Bluetooth classic to Bluetooth low energy is in the

way that a connectionless model has been designed and implemented. In a connectionless model, devices do not need to maintain a connection for useful information to be exchanged quickly between them. Because the main protocols never establish a connection-oriented channel between devices, there is no cost to dropping and then reconnecting a connection when data needs to be sent. This encourages devices to only establish a connection when they need to send data, and not to maintain an expensive connection just in case some data does need to be sent. This connectionless model does impose some interesting design changes from standard wireless protocols.

In a connection-oriented channel, the state information can be established over a period of time by using the protocol. The state information, therefore, typically is not available whenever it is required, but only by remembering the state that has been implicitly created by both devices. This state information requires a long time to be established, causing delays upon the initial connection while the state information is discovered and negotiated. Protocols that are based on implicit state typically have negotiation and configuration procedures as well as feature bits and version numbers. If a connection is going to be up for a long time, and there is a lot of state information, that state-full system can be more efficient.

Unfortunately, many protocols are not fully defined, with each bit of state implicitly defined as opposed to being explicitly defined. This leads to interoperability problems because each device thinks that the connection has a different state and therefore makes different assumptions about what can or should happen next. This is one of the biggest problems with connection-oriented systems. This can be solved by defining the state explicitly and also how any state machines work. A good example of this would be the Logical Link Control and Adaptation Protocol (L2CAP) Layer in Bluetooth where for Bluetooth classic, a simple state machine and configuration system are used when establishing a connection. All the state of the connection is explicitly defined, and the connection state machine is fully described. This, however, has taken over 10 years to develop to the exemplary level it now occupies.

Thus, the connectionless model solves these problems by not defining the state of a connection, but the state of the device. By exposing state through a stateless protocol, such as Attribute Protocol, it is possible to disconnect at any time and, upon reconnection, determine what the current state is directly from the other device. It is also possible to explicitly define state machines with both an exposed state and an exposed control point to persuade that state machine into different states as defined by some service. It is also possible to reestablish a connection just because some information in this state has changed and a device has registered to receive this state change information.

For example, it can be used to signal the battery level of a device. A monitoring device would connect to the battery-powered device, read the current battery level, configure the battery level to be notified when it changes, and then disconnect. When the battery level does change, the battery-powered device slowly makes itself connectable and the battery-monitoring device notices that it has something to say.

The battery-powered device then connects to the battery-monitoring device. Immediately after establishing a connection, the battery-powered device can notify the monitor of the new battery level and then immediately terminate the link. This can all happen within about 3 milliseconds. For a device that was fully charged and has a battery that lasts for just one year, this would require approximately 99 reports of 3 milliseconds each, a total radio-active time of just fewer than 300 milliseconds. In contrast, just to set up a connection-oriented channel in Bluetooth classic can take a similar time, *for each report*.

4.4. Gateways

The most radical change in computing technology over the last few years has been the spread and pervasiveness of the Internet. It appears that everything is connected to it, from newspapers to televisions and radios. All of these are portals for media, whether it is printed words, moving pictures, or voice. Of course, there are many other uses for the Internet, including communications such as e-mail and social networking, and teleconferencing audio and video links. However, the next big challenge will be to connect hundreds more devices for each device that's connected to the Internet today. This is a big change and the current infrastructure will probably not have the capacity to cope with all this new data initially.

The problem with the Internet is that it is built around a connection-oriented model. A TCP[1] connection is a session-oriented channel that is established between two devices, which takes time to set up. The fact that devices connected to the Internet have to have an address is also session based. To obtain an address, a device must either have this address programmed into it or it has to ask another device to allocate it an address for a period of time; it can use this address for this period of time, after which it must ask for another address.

The biggest problem with the Internet as it is currently structured is that it is designed around a wired infrastructure. Wires are great; they are mostly reliable, and because the devices connected with wires aren't moving around, they can also be connected to other wires supplying electricity. This means that energy efficiency for wired protocols is rarely considered. The fact that routers constantly check for the mapping of allocated or nonallocated Internet addresses to devices, at stochastic intervals, typically means that Internet devices need to be listening all the time. This doesn't work when devices are constantly moving around and need ultra-low power consumption. Another approach has to be used.

The model followed in Bluetooth low energy is one that is used in most homes that have more than one computer connected to the Internet. This is the concept of a gateway using network address translation (NAT). To the outside, your typical home has a single Internet address, allocated to the gateway or router. The gateway, however, allocates a separate set of addresses for all the devices in the home that are attached to it. The key is that the gateway translates the internal addresses to the single external address, hiding the topology of the internal network from the outside world. The outside world just sees one device, and it doesn't really care

about which device is really sending or receiving the data.

The Internet Protocol itself is very expensive. For an IPv6-based network, a 128-bit source and destination address has to be included in every single packet that is transmitted. This means that the minimum size of a packet, before any other protocol overhead, is 32 bytes. This is larger than the biggest Bluetooth low energy packet. Therefore, it becomes very difficult to just use the Internet Protocols directly over low energy—even if we were to discount the fact that they were designed when everything was wired. The gateway model, however, allows us to hide the internal addressing of devices from the outside world. This internal addressing of devices could be using a separate IPv6 address space or could be using some other addressing scheme that is transparent to the outside world.

By using Bluetooth low energy gateways, tiny wireless devices not only can connect to the Internet, but they can do so using the least possible amount of power. Bluetooth low energy does this by pushing the complexity of the Internet to the gateway devices that have the resources to cope. It also allows these gateway devices to map Internet addresses to devices using any scheme they want. This could be done by allocating an individual IPv6 address to each device or by using a port number of a single Internet address to each device.

Gateways are useful because if your refrigerator needs to notify its manufacturer that the compressor pump is failing and needs to be replaced under warranty, it must have a way to get this message out to the manufacturer's Internet server. Obviously, similar things will need to be done for all manufactured devices: washing machines, cars, and vacuum cleaners, to name just a few. The manufacturer might also want to send information to these devices; for example, they might want to upgrade washing programs. Thus, gateways provide the way for devices to interact with the Internet without being burdened by the power-hungry wired protocols that drive the Internet.