Bluetooth: Technology for Short-Range Wireless Apps

Pravin Bhagwat • *Reefedge Inc.*

Handheld devices are rapidly becoming an integral part of our daily lives, and many road warriors already carry a cell phone, palmtop, and laptop computer with them. In most cases, these devices do not have compatible data communication interfaces, or, if they do, the interface requires cumbersome cable connections and configuration procedures. An obvious solution is to get rid of the cables and use short-range wireless links to facilitate on-demand connectivity among devices. An ideal solution would also be inexpensive, enabling of compelling applications, and universally adopted by device vendors.

In 1998, five major companies (Ericsson, Nokia, IBM, Toshiba, and Intel) formed a group to create a license-free technology for universal wireless connectivity in the handheld market. The result is Bluetooth, a technology named after a 10th-century king who brought warring Viking tribes under a common rule. The Bluetooth specifications,[1,2] currently in version 1.1, define a radio frequency (RF) wireless communication interface and the associated set of communication protocols and usage profiles.

The link speed, communication range, and transmit power level for Bluetooth were chosen to support low-cost, power-efficient, single-chip implementations of the current technology. In fact, Bluetooth is the first attempt at making a single-chip radio that can operate in the 2.4-GHz ISM (industrial, scientific, and medical) RF band. While most early Bluetooth solutions are dual chip, vendors have recently announced single-chip versions as well. In this overview of the technology, I will first describe the lower layers of the Bluetooth protocol stack. I will also briefly describe its service discovery protocol and, finally, how the layers of the protocol stack fit together from an application's point of view.

Bluetooth Specifications

The Bluetooth 1.1 specification was released in February 2001. The specification consists of two parts: core and profiles.

Core Specifications

The core specification defines all layers of the Bluetooth protocol stack.[1] As shown in Figure 1, the Bluetooth stack differs from the classical seven-layer networking model in some ways. These differences are primarily to support ad hoc connectivity among participating nodes, while conserving power and accommodating devices that lack resources to support all layers of the classical networking stack.

The radio is the lowest layer. Its interface specification defines the characteristics of the radio front end, frequency bands, channel arrangements, permissible transmit power levels, and receiver sensitivity level. The next layer is the baseband, which carries out Bluetooth's physical (PHY) and media access control (MAC) processing. This includes tasks such as device discovery, link formation, and synchronous and asynchronous communication with peers. Bluetooth peers must exchange several control messages for the purpose of configuring and managing the baseband connections. These message definitions are part of the link manager protocol (LMP). The functional entity responsible for carrying out the processing associated with LMP is called the *link manager*.

Bluetooth is unique in offering the front-end RF processing integrated with the baseband module. On-chip integration lowers the cost of the network

 1089-7801/01/$10.00©2001 IEEE

MICROSOFT CORP.
EXHIBIT 1039

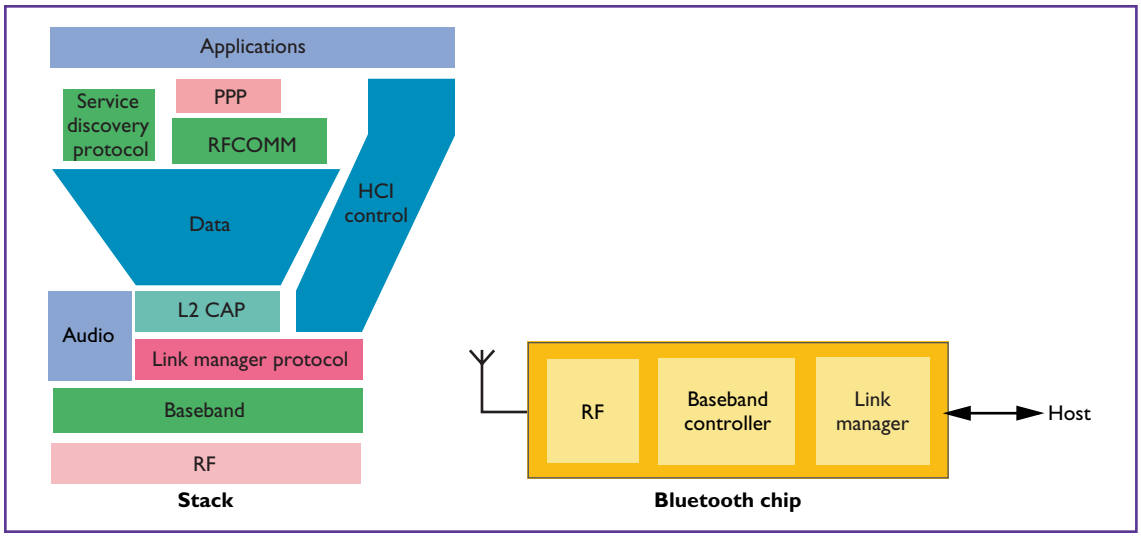

Figure 1. The Bluetooth networking stack and chip. The design supports the integration of an analog radio front end, signal-processing elements, and baseband controller on a single chip.

interface, and the small size makes it easy to embed Bluetooth chips in devices such as cell phones and PDAs. A Bluetooth chip can be connected to its host processor using USB, UART, or PC-card interfaces.

The Host Controller Interface (HCI) specification defines a standard interface-independent method of communicating with the Bluetooth chip. The software stack on the host processor communicates with the Bluetooth hardware using HCI commands. Since no hardware-specific knowledge is needed, the Bluetooth stack software can easily be ported from one Bluetooth chip to another. The HCI layer is part of the Bluetooth stack, but it doesn't constitute a peer-to-peer communication layer since the HCI command and response messages do not flow over the air link.

The logical link control and adaptation protocol (L2CAP) specification can be viewed as Bluetooth's link layer. Usually, L2CAP and layers above it are implemented in software. L2CAP delivers packets received from higher layers to the other end of the link. Bluetooth devices can establish an L2CAP connection as soon as they are in range of each other. A client device then needs to discover the services provided by the server device.

The service discovery protocol (SDP) defines the means by which the client device can discover services as well as their attributes. The SDP design has been optimized for Bluetooth. It defines only the discovery mechanisms; the methods for accessing those services are outside its scope.

The RFCOMM specification defines a method of emulating the RS-232 cable connection on top of the Bluetooth airlink. RFCOMM supports legacy applications that use the COM port to communicate with the peer host. For example, point-to-point (PPP) protocols expect a serial line interface from the lower layer. Since PPP provides a packet-oriented interface to the higher layers, all packet-based network and transport protocols, including TCP/IP, can be supported on top of PPP. More efficient methods of running IP over Bluetooth are currently under development.

Profile Specifications

Vendors can use the services offered by the Bluetooth stack to create a variety of applications. Because interoperability is crucial to Bluetooth's operation, the Bluetooth SIG has defined profile specifications to support it.[2] The current specifications include 13 profiles listed in Table 1 (next page).

The profiles specify controller and stack parameter settings as well as the features and procedures required for interworking among Bluetooth devices. All vendor implementations of these profiles are expected to be interoperable. The Bluetooth certification authority uses the profiles to test and certify compliance, and grants use of the Bluetooth logo only to products that conform to the methods and procedures defined in the profiles.

Radio Front End

The 2.4-GHz ISM band in which Bluetooth operates is globally available for license-free use. Europe and the United States allocate 83.5 MHz to this band, but Spain, France, and Japan allocate less. To accommodate these differences, 79 channels spaced 1 MHz apart are defined for Europe and the U.S., and 23 RF channels spaced

Table 1. Profiles defined in Bluetooth 1.1 specifications.

Use case	Description
Generic access	Generic procedures for discovery and link management of connecting to Bluetooth devices.
Service delivery	Features and procedures for a Bluetooth device application to discover services registered in other devices.
Cordless telephone	Features and procedures for interoperability between different units active in a "3-in-1"phone.
Intercom	Requirements for supporting intercom functionality within a "3-in-1" phone.
Serial port	Requirements for setting up emulated serial cable connections using RFCOMM between two peer devices.
Headset	End-user service requirements and interoperability features for Bluetooth devices implementing headsets.
Dial-up networking	End-user service requirements and interoperability features for Bluetooth devices implementing dial-up networking.
Fax	End-user service requirements and interoperability features for Bluetooth devices implementing fax services.
LAN access	Definition of (a) how Bluetooth devices can access LAN services using PPP and (b) how the PPP mechanisms form a network.
Generic object exchange	Requirements for Bluetooth devices to support object exchange usage models.
Object push	Application requirements for Bluetooth devices to support the object push usage model.
File transfer	Application requirements for Bluetooth devices to support the file transfer usage model.
Synchronization	Application requirements for Bluetooth devices to support the synchronization usage model.

1 MHz apart are defined for Spain, France, and Japan. Efforts are under way to open up the full width of the spectrum in Spain and France, as well as in Japan so that Bluetooth devices would function worldwide.

Bluetooth is a frequency-hopping spread-spectrum system. This means that the radio hops through the full spectrum of 79 or 23 RF channels using a pseudorandom hopping sequence. The hopping rate of 1,600 hops per second provides good immunity against other sources of interference in the 2.4-GHz band. The link speed is 1 Mbps, which is easily achieved using a simple modulation technique (Gaussian Frequency Shift Keying, or GFSK). A more complex modulation technique could achieve a higher rate, but GFSK keeps the radio design simple and low cost.

The radio front end is usually the most costly part of a wireless network interface. In typical radio receivers, the RF filters, oscillators, and image-reject mixers process input signals at high frequencies. Such circuits require expensive materials. To keep costs down, Bluetooth recommends shifting the input signal to a lower intermediate frequency (IF, around 3 MHz), which allows on-chip construction of low-power filters using CMOS material. Shifting to low IF, however, creates new problems, such as reduced receiver sensitivity. Recommended receiver sensitivity for Bluetooth is –70 dBm or better. The comparable number for IEEE 802.11 Wireless LANs is about –90 dBm). Thus, for the same transmit power, the range for Bluetooth is shorter than it is for 802.11 WLAN.

Piconets and Scatternets

A set of Bluetooth devices sharing a common channel is called a *piconet*. As shown on the left side of Figure 2, a piconet is a star-shaped configuration in which the device at the center performs the role of *master* and all other devices operate as *slaves*. Up to seven slaves can be active and served simultaneously by the master. If the master needs to communicate with more than seven devices, it can do so by first instructing active slave devices to switch to low-power park mode and then inviting other parked slaves to become active in the piconet. This juggling act can be repeated, which allows a master to serve a large number of slaves.

Most envisioned Bluetooth applications involve local communication among small groups of devices. A piconet configuration consisting of two, three, or up to eight devices is ideally suited to meet the communication needs of such applications. When many groups of devices need to be active simultaneously, each group can form a separate piconet. The slave nodes in each piconet stay synchronized with the master clock and hop according to a channel-hopping sequence that is a function of the master's node address. Since channel-hopping sequences are pseudorandom, the probability of collision among piconets is small. Piconets with overlapping coverage can coexist and operate independently. Nonetheless, when the degree of overlap is high, the performance of each piconet starts to degrade.

In some usage scenarios, however, devices in different piconets may need to communicate with

each other. Bluetooth defines a structure called *scatternet* to facilitate interpiconet communication. A scatternet is formed by interconnecting multiple piconets. As shown on the right side of Figure 2, the connections are formed by *bridge nodes*, which are members of two or more piconets. A bridge node participates in each member piconet on a time-sharing basis. After staying in a piconet for some time, the bridge can turn to another piconet by switching to its hopping sequence. By cycling through all member piconets, the bridge node can send and receive packets in each piconet and also forward packets from one piconet to another.

A bridge node can be a slave in both piconets or be a slave in one and a master in another.[3] For example, consider a room full of people, where each person has a cell phone and a cordless headset. When users speak into their headsets, only the cell phones paired with their headsets should pick up the signal. In this example, each headset and cell phone pair constitutes a separate piconet. Now suppose these users also want to send text messages from their cell phones to one another. This will be possible only if all piconets are interconnected to form a large scatternet.

The techniques for forming scatternets are still under development.[4]

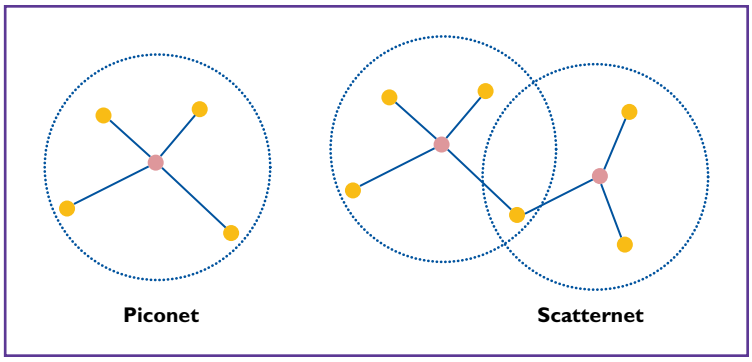

Figure 2. Piconet (left) and scatternet (right). The master device at the center of a piconet can serve up to seven slaves; members of two or more piconets are called bridge nodes, which support interpiconet communication.

Inquiry and Paging

Bluetooth uses a procedure known as *inquiry* for discovering other devices; it uses *paging* to subsequently establish connections with them. Both inquiry and paging are asymmetric procedures. In other words, they involve the inquirer and the inquired (as well as the pager or the paged) devices to perform different actions. This implies that when two nodes set up a connection, each needs to start from a different initial state; otherwise, they would never discover each other. The profile specifications play an important role here, defining the required initial state for each device in all usage scenarios. A symmetric procedure for establishing connections is an ongoing topic of research.[4]

The inquiry and paging are conceptually simple operations, but the frequency-hopping nature of the physical layer makes the low-level details quite complex. Two nodes cannot exchange messages until they agree to a common channel-hopping sequence as well as the correct phase within the chosen sequence. Bluetooth solves this problem simply by mandating the use of a specific inquiry-hopping sequence known to all devices. During inquiry, both nodes (one is the listener and the other is the sender) hop using the same sequence; but the sender hops faster than the listener, transmitting a signal on each channel and listening between transmissions for an answer. When more than one listener is present, their replies may collide. To avoid the collision, listeners defer their replies until expiration of a random backoff timer. Eventually the sender device collects some basic information from the listeners, such as the device address and the clock offsets. This information is subsequently used to page the selected listener device.

The communication steps during the paging procedure are similar, except that the paging message is unicast to a selected listener, so the listener need not back off before replying. The sender also has a better estimate of the listener's clock, which enables it to communicate with the listener almost instantaneously. Upon receiving an ACK for the paging message, the sender becomes the master and the listener becomes the slave of the newly formed piconet, and both nodes switch to the piconet's channel-hopping sequence. Later, if necessary, the master and slave roles can be swapped.

The steps for admitting a new slave into an existing piconet are slightly more complex. The master can either start discovering new nodes in its neighborhood and invite them to join the piconet or, instead, wait in scan (listen) state and be discovered by other nodes. With both options, communication in the original piconet must be suspended for the duration of the inquiry and paging process. The latency of admitting a new node into the piconet can be large if the master does not switch to the inquiry or scan modes frequently. This latency can be reduced only at the cost of some piconet capacity. The study of this trade-off is another topic of ongoing research.

Table 2. Channel throughput for different packet sizes.

Packet size (in slots)		Throughput in Kbps (with FEC)		Throughput in Kbps (no FEC)	
In slave direction	In master direction	In slave direction	In master direction	In slave direction	In master direction
1	1	108.8	108.8	172.8	172.8
3	1	387.2	54.4	585.6	86.4
5	1	477.8	36.3	723.2	57.6

Low-Power Modes

Bluetooth offers different low-power modes for improving battery life. Piconets are formed on demand when communication among devices is ready to take place. At all other times, devices can be either turned off or programmed to wake up periodically to send or receive inquiry messages. When a piconet is active, the slaves stay powered on to communicate with the master. It is possible to switch a slave into a low-power mode whereby it sleeps most of the time and wakes up only periodically.

Three types of low-power modes have been defined:

- *Hold mode* is used when a device should be put to sleep for a specified length of time. As described earlier, the master can put all its slaves in the hold mode to suspend activity in the current piconet while it searches for new members and invites them to join.
- *Sniff mode* is used to put a slave in a low-duty cycle mode, whereby it wakes up periodically to communicate with the master.
- *Park mode* is similar to the sniff mode, but it is used to stay synchronized with the master without being an active member of the piconet. The park mode enables the master to admit more than seven slaves in its piconet.

Piconet Channel

As soon as a piconet is formed, communication between the master and the slave nodes can begin. The piconet channel is divided into 625-microsecond intervals, called slots, where a different hop frequency is used for each slot. The channel is shared between the master and the slave nodes using a frequency-hop/time-division-duplex (FH/TDD) scheme whereby master-slave and slave-master communications take turns. Slave-to-slave communication is not supported at the piconet layer. If two slaves need to communicate peer to peer, they can either form a separate piconet or use a higher layer protocol, such as IP over PPP (see Figure 1), to relay the messages via the master.

At a 1-Mbps link speed, a 625-microsecond slot time is equivalent to the transmission time of 625 bits. However, a single slot packet size in Bluetooth is only 366 bits. This reserves enough guard time to let the frequency synthesizers hop to the next channel frequency and stabilize. Discounting space for the headers leaves 30 bytes for the user payload.

Synchronous Link

To transmit real-time voice, an application must reserve a slot in both directions at regular intervals. In Bluetooth terminology, this is called a synchronous (SCO) link. An SCO link can transport telephone-grade voice. The speech coder generates 10 bytes every 1.25 milliseconds. Since a baseband packet can carry up to 30 bytes in each slot, only one slot in each direction is needed every 3.75 ms (or every sixth slot). The packet type that carries 30 voice bytes is called an *HV3 packet*. This packet is transmitted without coding or protection, and is not retransmitted if it is lost.

To cope with bit errors when the channel conditions are not perfect, some forward error correction (FEC) should be added to the voice payload. An HV2 packet carries 20 bytes of voice plus 10 bytes of redundant data (2/3 FEC code). Since 20 bytes of speech is generated in 2.5 ms, the SCO link should reserve one slot in each direction every 2.5 ms (or every fourth slot). To cope with extreme channel conditions, the baseband specification also defines an HV1 packet that carries only 10 bytes of speech and 20 bytes of FEC code. An HV1 SCO link uses up the entire channel capacity. This means that all data transfer sessions will be suspended when an HV1 SCO connection is in progress.

Asynchronous Link

Data communication between a master-slave pair involves a different set of considerations. For example, the data payload must be protected by a cyclic redundancy check (CRC) so that the receiver can determine whether the received bits are in error. When losses occur, the baseband layer should retransmit the data. Furthermore, to make efficient use of the piconet channel, slots should be allocated on demand, instead of being reserved for the usage duration. A data path between a master-slave pair meeting all of these requirements is called an asynchronous data link (ACL). SCO links have pri-

ority over data, so ACLs can claim only unused slots. Only a single ACL can exist between a master and a slave.

The master is responsible for distributing available slots among all ACLs. This scheme has two advantages:

- the master can ensure that the slave transmissions do not collide; and
- the slots can be allocated to satisfy the quality of service (QoS) requirement of each ACL. The master can grant more bandwidth to a slave by polling it more frequently or by changing the packet size.

The baseband specification does not mandate the use of any specific slot-allocation scheme. Chip vendors can choose any policy that fits their target applications.

As with SCO packets, the payload size of single-slot ACL packets is limited to 30 bytes. After discounting space for the higher layer headers and the CRC, only 27 bytes are left to transport application data. When FEC is added, the available space goes down to 17 bytes. To improve channel efficiency, the baseband specification has defined multislot packets, which are three or five slots long and transmitted in consecutive slots. The transmitter stays fixed on a hop frequency during the length of packet transmission and skips over the missed hops after the transmission is complete. This reduces the effective channel-hopping rate, but increases the channel efficiency because of fewer hops.

Table 2 shows the achievable throughput in the master-to-slave and slave-to-master directions as a function of packet size, with and without FEC. Although link speed is 1 Mbps, achievable aggregate throughput can range from 217.6 Kbps to 780.8 Kbps. The presence of an HV3 or HV2 SCO link significantly reduces the achievable throughput of an ACL.

Logical Link Control and Adaptation Protocol

L2CAP can be viewed as the data plane of the Bluetooth link layer (see Figure 3). Because the baseband packet size is too small for transporting higher layer packets, a thin layer is needed for exporting a bigger packet size to the higher layers. While a number of generic segmentation and reassembly protocols could be used or adapted for use over ACLs, the Bluetooth SIG instead defined L2CAP, which is highly optimized to work in conjunction with the baseband layer. For example,

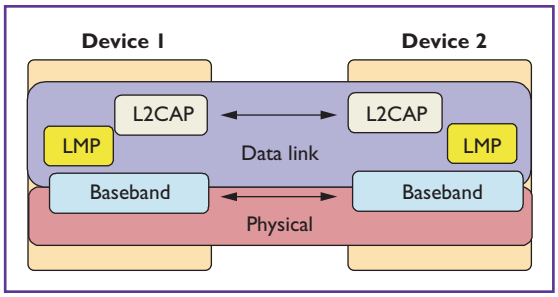

Figure 3. Lower part of stack. L2CAP can be viewed as the data plane of Bluetooth's link layer.

L2CAP does not support integrity checks because the baseband packets are already CRC protected. Likewise, it is assumed that the lower layer delivers packets both reliably and in sequence. These two assumptions significantly simplify the design of segmentation and reassembly logic. The only caveat is that L2CAP will not work if used over any media other than the Bluetooth baseband.

The multiplexing and demultiplexing of higher layer protocols is supported using channels, multiple instances of which can be created between any two L2CAP endpoints. Each higher layer protocol or data stream is carried in a different channel. The L2CAP channels are connection oriented in the sense that they require an explicit phase to establish the channel, during which both ends choose a local name (channel identifier) and communicate it to the other end. Subsequently, each packet sent over the channel is tagged with the channel identifier, which—within the context of the receiver—uniquely identifies the source as well as the protocol being transported over the channel.

The L2CAP specification also defines a connectionless channel for supporting broadcast and multicast group communication, but this feature is not yet fully developed.

Service Discovery Protocol

Both ends of a Bluetooth link must support compatible sets of protocols and applications to successfully exchange data. In some cases it may also be necessary to configure protocol and stack parameter settings before applications can be started. Such configuration settings cannot be chosen statically, since some parameters may require adjustment to match the features and services supported by the peer Bluetooth device.

Bluetooth's SDP provides a standard means for a Bluetooth device to query and discover services supported by a peer Bluetooth device. SDP is a client-server protocol. The server maintains a list of service records, which describe the characteris-

tics of services hosted at the server. By issuing SDP queries, a client can browse all available service records maintained at the server or retrieve specific attribute values from a service record.

In addition to defining query and response protocol formats, the SDP specification also defines a standard method for describing service attributes. Service attributes are represented using an <identifier, value> pair. The 1.1 specification defines some of the commonly used services, but developers have the freedom to define new subclasses of the standard services or to create new services on their own.

Since new service definitions do not require any coordination with the Bluetooth SIG numbering authority, it is necessary to ensure that two independently created service definitions do not conflict. Collisions are avoided by associating each service definition with a universally unique identifier (UUID) which is generated once at the time a service is defined. UUIDs of the services defined by the Bluetooth SIG are included in the assigned numbers document.

If the client already knows the UUID of the service it is looking for, it can query the SDP server for specific service attributes. Alternatively, the client can browse the list of available services and select from the list. These are the only two search options supported in SDP. Although other IP-based service discovery protocols, such as SLP and Jini, provide richer service description schema and more powerful search capabilities, the Bluetooth SDP has two advantages:

- The majority of version-1.1-compliant Bluetooth devices will be non-IP devices. Requiring them to support IP only for the sake of supporting SLP would be costly.
- SDP is optimized to run over L2CAP. Its limited search capabilities and non-text-based attribute-id and attribute-value descriptions lend an efficient and small footprint implementation for small devices.

SDP provides a mechanism only for retrieving service information from other devices. Methods of invoking those services are outside the scope of SDP.

Link Manager Protocol

Before a device can establish the L2CAP channel, the link manager must carry out a number of baseband-specific actions, such as piconet creation, master-slave role assignments, and link configuration. These functions belong to the control plane of the Bluetooth link layer and require the link manager to exchange LMP messages over the air link. Depending on the operating environment, the link manager must adjust a number of piconet and link-specific parameters. For example, the peer-link controller can be instructed to switch to a low-power mode, adjust its power level, increase the packet size, and change the requested QoS on an ACL.

Security can also be configured using LMP messages. Before a data or voice exchange can begin, Bluetooth devices should be able to authenticate each other. Likewise, transmission over the air link must be encrypted to provide protection from eavesdroppers. Both objectives are easy to achieve when a security association already exists between a pair of devices. The link manager can use the shared secret key to verify the peer device's authenticity as well as to negotiate a link key for encryption. A typical session between two Bluetooth devices begins with the formation of a piconet, followed by the exchange of LMP messages first to authenticate and then to negotiate new encryption keys with the peer device. Only upon successful completion of the LMP handshake can further data exchange or voice communication take place.

The level of security built into the version 1.1 specifications is satisfactory so long as the initial security associations are computed in a secure fashion. The baseband and LMP specifications also define a method, called *pairing*, for creating a new security association between two devices when they pair for the first time. The method uses an out-of-band channel for creating a security association, which is then used as a seed to compute a cryptographically secure shared secret key. By out-of-band channel I mean a user typing a randomly chosen PIN number on both devices. Clearly, the security of a pairing phase is limited by a user's ability to choose good PIN numbers. In scenarios when one device in the pair does not have a keypad, security can be further compromised if the chosen PIN is transmitted to the other device in clear text.

Putting the Pieces Together

The ultimate objective of the Bluetooth specifications is to allow multivendor applications to interoperate. Different applications may run on different devices, and each device may use a protocol stack from one vendor and a Bluetooth chip from another. Yet interoperability among applications is achieved when different implementations comply with the same core and profiles specifications.

At the lowest layer, Bluetooth chips from different vendors interoperate over the air link because all Bluetooth chips implement the baseband and LMP specifications. Bluetooth stacks, which can be implemented as either firmware or software, include the L2CAP, SDP, and RFCOMM layers. It is relatively easy to port a Bluetooth stack from one platform to another because the lowest layer of a Bluetooth stack interfaces with a Bluetooth chip via a standard HCI interface which is also a part of the 1.1 specifications.

Porting a Bluetooth application from one stack to another, however, is more difficult. The application can use any standard API to access IP, PPP, OBEX, or RFCOMM layers of the Bluetooth stack, but there is no standard API to access the control functions provided by the Bluetooth stack. For example, if an application were to initiate a Bluetooth inquiry to discover other devices in its neighborhood, it must use an API specific to the stack vendor to access those functions.

Support for RFCOMM has been provided only for backward compatibility reasons. Legacy applications that run over serial cable, such as OBEX and PPP, will work over any Bluetooth stack without modifications. Thus, synchronization and IP-based applications already developed by vendors can be made available immediately when PDAs, cell phones, and laptops are Bluetooth enabled. The next release of Bluetooth specifications will provide better support for IP (without going through the PPP and RFCOMM layers), thus increasing portability of IP-based applications across all Bluetooth platforms. Standardization of control APIs, however, remains an unfinished task that has not yet been taken up by any standards organization.

Conclusion

Whether Bluetooth will live up to its promise or not will depend on a number of factors, some of which involve market forces rather than technical issues. For example, unless the initial adoption of Bluetooth is high, it will be difficult to meet the low-cost objective.

Security is also an open issue—as it is in almost all Internet applications. The free flow of information is desirable in some scenarios, but in general, proper safeguards are required to prevent the unauthorized leakage of information. The Bluetooth SIG is addressing the security issues associated with the initial usage scenarios, but new applications of Bluetooth will require a closer look at potential security threats.

One technical issue is that profile-based interoperability is easy to manage when the number of profiles is small, but market predictions indicate that more than a billion devices will be equipped with Bluetooth chips by 2005. This number is significantly greater than the number of hosts connected to the Internet today. As people find innovative uses of this technology, new profiles will be needed. Ensuring compliance with a rapidly increasing number of profiles will likely be difficult to maintain in the future. A good solution to this problem would be to quickly standardize an IP-over-Bluetooth specification since interoperability at the IP layer would automatically translate into interoperability at the applications layer. Some efforts have already begun within the Bluetooth SIG as well as in the IETF to resolve this issue.

Bluetooth has caught the attention of consumers because it would enable them do things that are otherwise cumbersome or not possible: synchronizing data between cell phones, laptops, and PDAs; using cell phones as cordless phones when at home; and connecting PDAs to the office LAN. The value proposition is therefore strong. The challenge is for vendors to meet these expectations.

References

1. *Specification of the Bluetooth System — Core*; available online at http://www.bluetooth.com/developer/specification/Bluetooth_11_Specifications_Book.pdf.
2. *Specification of the Bluetooth System — Profiles*; available online at http://www.bluetooth.com/developer/specification/Bluetooth_11_Profiles_Book.pdf.
3. G. Miklos et al., "Performance Aspects of Bluetooth Scatternet Formation," poster presentation at Mobile Ad Hoc Networks and Computing (MobiHOC 2000), IEEE/ACM workshop, Aug. 2000.
4. T. Salonidis et al., "Distributed Topology Construction of Bluetooth Personal Area Networks," *Proc. IEEE Infocom 2001*, IEEE Communication Society, New York, 2001.

Pravin Bhagwat is the principal architect at Reefedge Inc., a networking infrastructure and software company that builds Bluetooth solutions for enterprise customers. He received a PhD in computer science from the University of Maryland, College Park. Bhagwat co-chaired the first Internet Engineering Task Force's BOF on IP over Bluetooth. He is chief architect of BlueSky, an indoor wireless networking system for palmtop computers, and co-inventor of TCP splicing, a technique for building fast application layer proxies.

Readers may contact the author at pravin@reefedge.com; http://www.cs.umd.edu/~pravin.