
Bluetooth

Part 3: Link Controller and JSR-82 API Architecture

Kjell Jørgen Hole

UiB



Last updated 24.02.09

Mail: Kjell.Hole@ii.uib.no

URL: www.kjhole.com

- Definition of *Link Control Protocol* (**LCP**)
- Acknowledgement/Request (ARQ) scheme
- Link controller states
- Link controller operation
- Piconet/Scatternet operation
- Master/Slave role switching
- Low-power operation
- High-level architecture of JSR-82 APIs

- LCP configures and controls baseband
- Responsible for packet-level access control, i.e., LCP determines what packet is going to be sent next
- Carries out higher level operations such as *inquiry* and *paging*
- Manages multiple links between different devices and piconets

... LCP Defined

KJhole.com

- LCP does not use its own packets. Instead, header bits in base-band packets, such as ARQN and SEQN, are used to signal between link controllers
- LCP is carried in the logical LC (Link Control) channel which is carried over SCO, eSCO, and ACL links

- The ARQN flag in the packet header indicates the status of the *previously* received packet:
 - ARQN = 1 (ACK) indicates that the packet was received correctly
 - ARQN = 0 (NAK) means that the previous receive failed

- The SEQN flag is toggled each time a *new* packet with a CRC is transmitted
- The SEQN flag remains the same for a retransmitted packet
- The receiver is able to tell the difference between a packet which has been retransmitted because
 - the receiver NAK'ed the previous packet
 - the transmitter failed to receive the ACK flag correctly

- If a packet is received correctly, but the SEQN flag is the same as the previously received SEQN flag, then the packet is ignored
 - An ACK is returned so that the transmitter will stop retransmitting
- When a Master broadcasts a packet to all its Slaves (LT_ADDR=0), the ARQN and SEQN flags are not used

Standby State in which Bluetooth device is inactive, radio not switched on, state used to enable low power operation

Inquiry State in which device tries to discover all Bluetooth enabled devices in the close vicinity:

- special fast hopping sequence used
- General or Dedicated Inquiry Access Code (GIAC, DIAC) used
- FHS packets with device information, such as CLNK and BD_ADDR, received from available devices
- list of all available devices is compiled

Inquiry Scan Most devices periodically enter the inquiry scan state to make themselves available to inquiring devices. Slow hopping sequence used

Page Master enters page state and starts transmitting paging messages to Slave using earlier gained access code and timing information

Page Scan Device periodically enters page state to allow paging devices to establish connections

Even More Link Controller States KJhole.com

Connection–Active Slave switches to Master's CLK and thus moves to the Master's frequency hop and timing sequence. Master transmits POLL packet to verify link, Slave sends a NULL packet

Connection–Hold Device ceases to support ACL traffic for a period of time, maintains LT_ADDR

Connection–Sniff Device listens in pre-defined time slots only

Connection–Park Device listens for traffic only occasionally, gives up its LT_ADDR

Link Controller Operation

KJhole.com

- LC operations between two devices defined by state diagram
- State transitions from standby into connection for a *single* link is illustrated in Figure 3-1

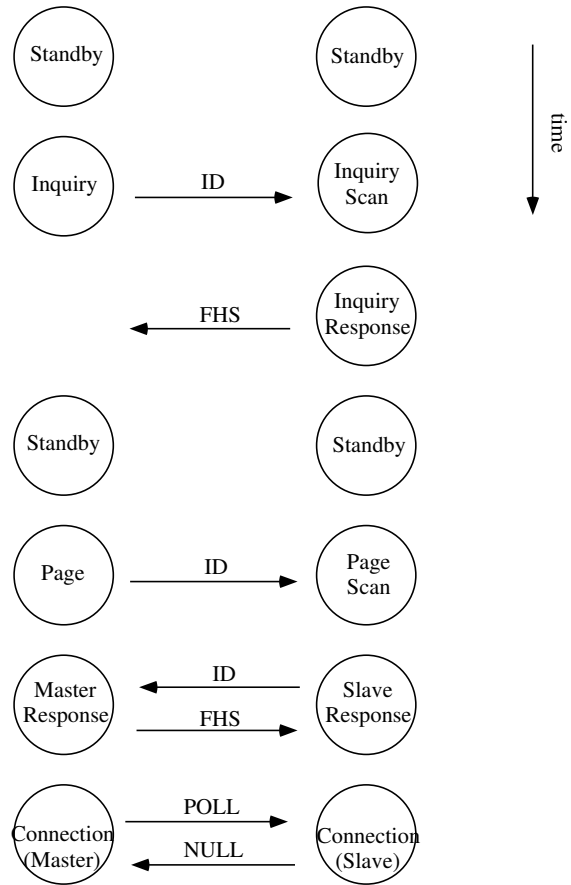


Figure 3-1 State transitions and packet exchanges as a pair of devices moves from standby into connection

Device Discovery and Inquiry

KJhole.com

1. *Inquiring device* sends ID packets with a GIAC or DIAC, device transmits twice per slot
2. *Inquiry scanning device* listens, hears the inquiry, waits for random back-off period, between 0 and 1023 slots, and re-enters inquiry scan state
3. Listens once more, hears inquiry, replies with FHS packet (no back-off period)

4. *Paging device* sends ID packets with Slave's access code, Master transmits twice per slot
5. *Page scanning device* listens, hears the inquiry, replies with ID packet containing Slave's access code (no need for random back-off delay)
6. Pager replies with FHS packet
7. Page scanner returns ID packet with Slave's access code

Connection Establishment

KJhole.com

Both devices move to paging device's hop sequence

8. *Master* sends POLL packet to *Slave* to check that the frequency hop sequence switch has happened correctly

9. *Slave* responds with NULL packet

Piconet Operation

KJhole.com

- The Master in a piconet will transmit to and receive from all Slaves which are allocated LT_ADDR addresses and are active
- If there is nothing to send to a Slave, the Master may either omit that Slave or transmit a NULL packet for timing purposes

ACL Links in Piconet

KJhole.com

- Master may communicate with several Slaves in sequence (see Figure 3-2)
- Master always transmits on even slots and the Slave replies on odd-numbered slots
- Slave can only transmit if it has been transmitted to

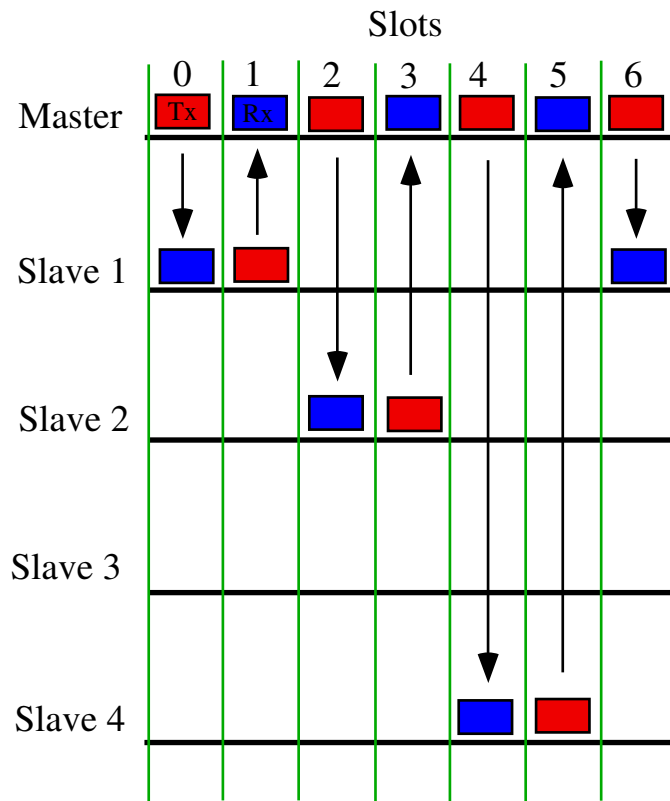


Figure 3-2 Piconet in operation with ACL traffic

- Master transmits on even numbered slot and Slave replies on the following odd numbered slot
- SCO slots may be spaced to use every slot pair, every second slot pair, or every third pair
- Only three SCO links can be supported in a piconet
- SCO links greatly reduce the opportunity to discover neighboring devices (!)

- A scatternet consists of two or more piconets, where it is possible to send information from any member device to any other member device
- A device which is active in two piconets must maintain and select between two piconet clocks, CLK_1 and CLK_2
- The switch in timebases requires guard time to synchronize to the new timebase

SCO Links in Scatternet

KJhole.com

- It is not possible for a device to have SCO links in two piconets at once because over time the piconet clocks will drift until eventually the two SCO links overlap

Master/Slave Role Switching

KJhole.com

- Any device may request a Master/Slave role switch with respect to another device it is communicating with
- A Master may temporarily become a Slave to allow a new device to be the Master and thus join a piconet quickly by paging
- A Slave which becomes a Master can acquire the former Master's Slaves (this acquisition is not supported by the HCI)

Uniting Scatternets with Role Switch (1)

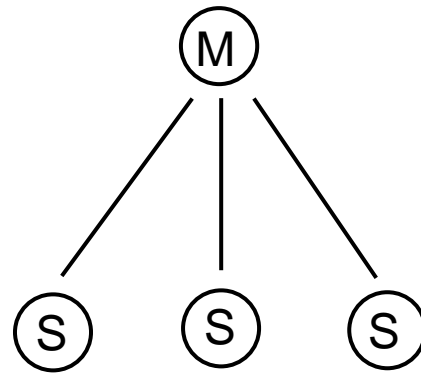


Figure 3-3 Piconet with Master and several Slaves

Uniting Scatternets with Role Switch (2)

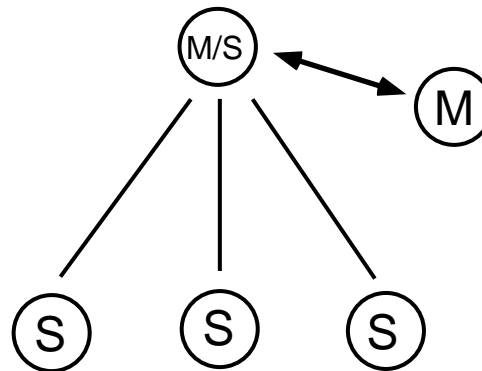


Figure 3-4 The piconet's Master page scans, allowing a new device to connect as its Master. The piconet's Master is now also a Slave; a Scatternet has been established

Uniting Scatternets with Role Switch (3)

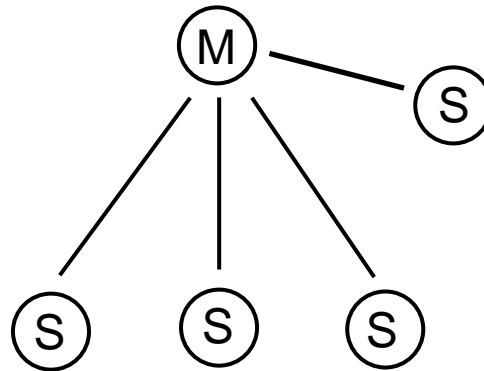


Figure 3-5 Master Slave switch on the new link. The scatternet is united into a piconet

- Many Bluetooth devices are battery powered. Bluetooth therefore provides the three low-power modes: *Hold*, *Sniff*, and *Park*
- The radio is the biggest power drain on a Bluetooth device, but the voltage controlled oscillator that drives the Bluetooth clock is another power hungry component which can be switched off
- Bluetooth provides the means to switch to a less accurate lower power oscillator when the accuracy of the normal oscillator is not needed

- A Bluetooth application may instruct the device it is running on to go into sleep mode for significant portions of its duty cycle
- Power management must be implemented such that it does not adversely affect the performance of the application
- Correct power management has the potential to extend the battery life of a Bluetooth device significantly

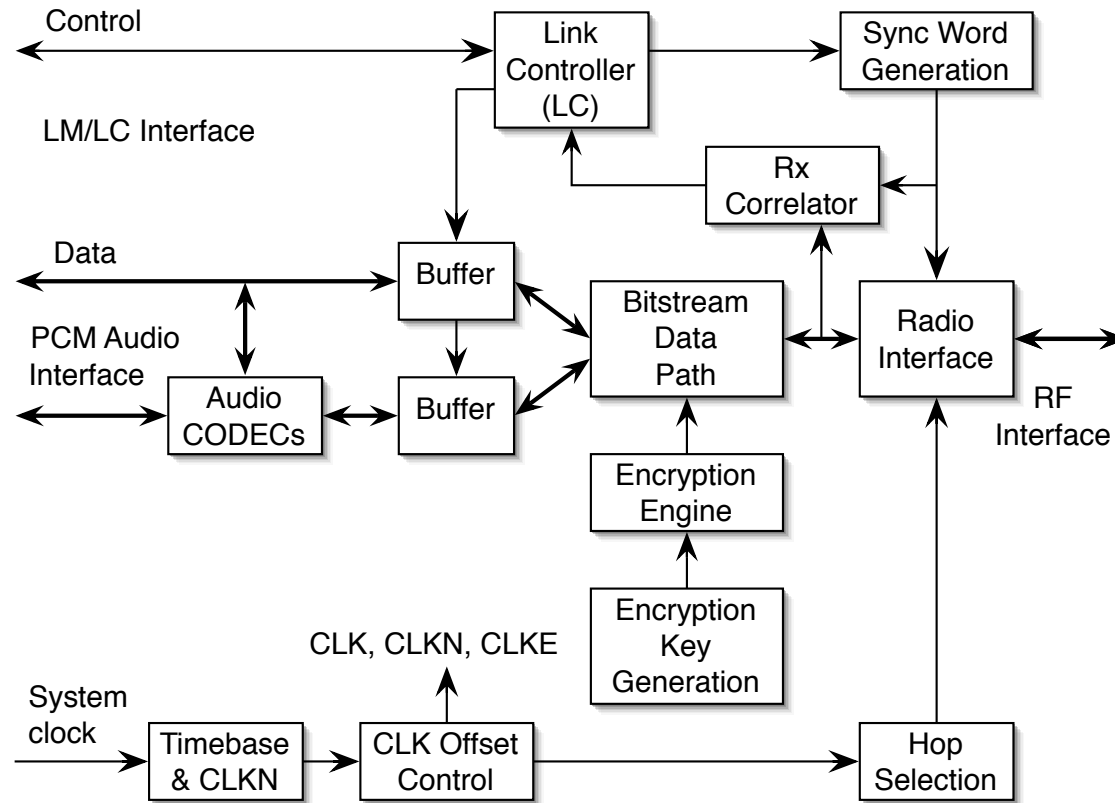


Figure 3-6 Baseband/LC architecture

Java ME

High-Level Architecture of JSR-82 APIs

The functionality provided by the JSR-82 APIs falls into three categories:

Discovery Device discovery, service discovery, and service registration

Communication Establishing RFCOMM, L2CAP, and OBEX connections

Device management Managing and controlling connections, including the security aspects

- The JSR-82 APIs only depend on a set of APIs known as the *Generic Connection Framework* (GCF)
 - GCF is a subset of the CLDC configuration
- JSR-82 is often coupled with the Java ME profile MIDP
- The OS contains the host part of the Bluetooth stack. Native Bluetooth applications interface with the OS directly
 - we will only consider applications that utilize the JSR-82 APIs

Architecture Diagram

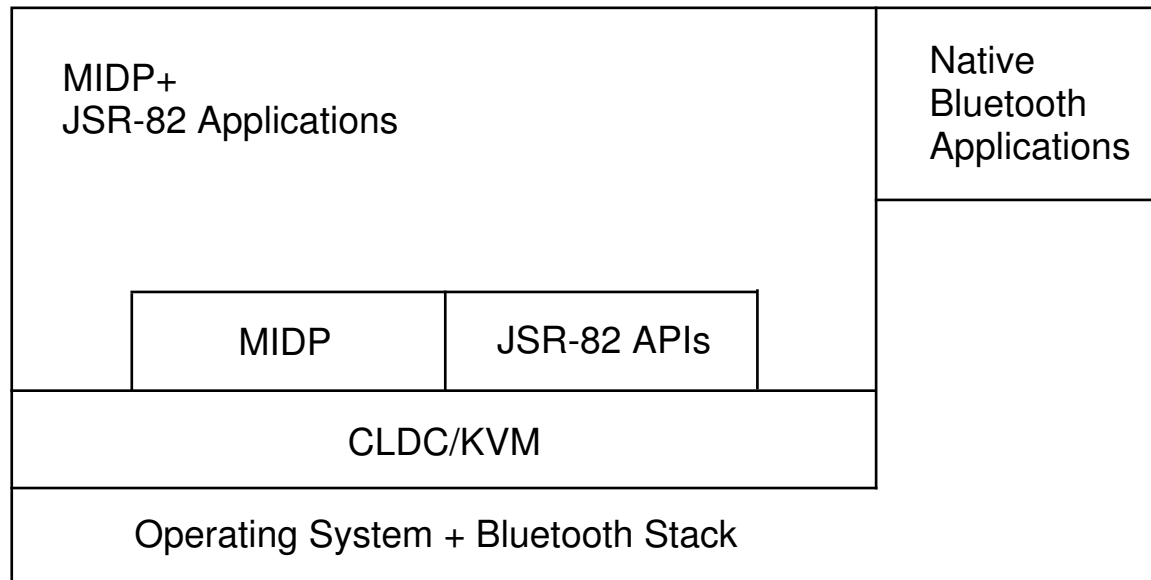


Figure 3-7 CLDC+MIDP+Bluetooth architecture diagram

- JSR-82 defines two separate packages:
 1. `javax.bluetooth`—requires Bluetooth stack
 2. `javax.obex`—independent of Bluetooth stack
- Both packages depend on the `javax.microedition.io` package which contains the GCF

Bluetooth client A user of services obtained via Bluetooth

Bluetooth server A provider of services via Bluetooth

- services are made available to remote clients by the definition of a service record describing the service
- a service record is added to the Service Discovery DataBase (**SDDB**) of the local device

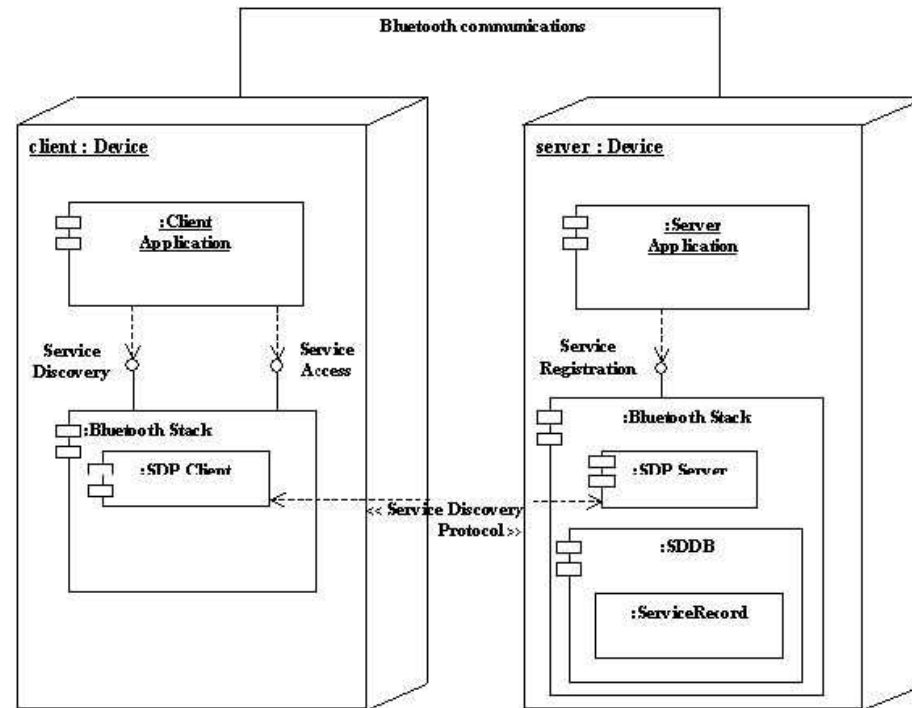


Figure 3-8 Bluetooth components involved in service discovery

- Create a service record
- Add service record to server's SDDB
- Register the Bluetooth security measures associated with service
- Accept connections from clients
- Update service record
- Remove or disable service record

Client Responsibilities

KJhole.com

- Use SDP to query a remote SDDB for desired services
- Initiate connections to servers offering desired services

Remark: The Java APIs also support peer-to-peer applications in which an application is capable of being both server and client

- The *Bluetooth Control Center* (**BCC**) allows multiple applications to run simultaneously without adversely affecting each other
 - no Java API provides direct access to the BCC
- The BCC performs three tasks:
 1. Resolving conflicting requests between applications
 2. Enabling modifications to the properties of the local device
 3. Handling security operations that may require user interaction

BCC Placement

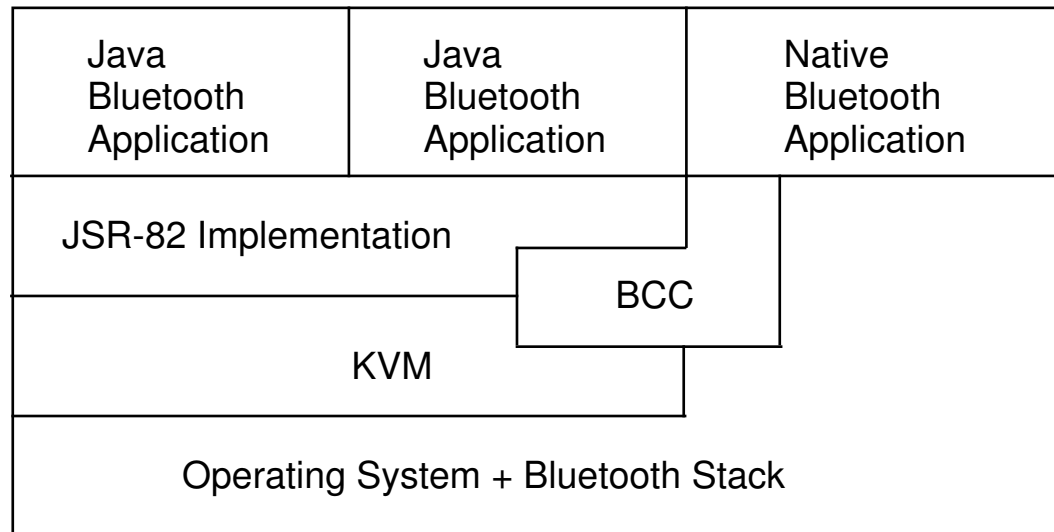


Figure 3-9 BCC and a JSR-82 implementation

More on BCC Tasks

KJhole.com

Conflict resolution Multiple applications may request different security settings on a Bluetooth link, or they may request to set the Bluetooth device into different discoverable modes. The BCC is responsible for resolving these conflicting requests

- LC layer is responsible for managing device discoverability, establishing connections, and once connected, maintaining the on-air links
 - it is up to a device to allow itself to be connected with
 - a Master and a Slave can exchange roles
 - three low-power modes are supported
- High-level architecture of JSR-82 APIs supports the client and server model, as well as the peer-to-peer model