

TACD

Lab session 3: Radio Communication

Maria J. Blesa



ALBCOM Research Group

**UNIVERSITAT POLITÈCNICA
DE CATALUNYA
BARCELONATECH**

About this session

- The goal is to get in touch with the radio communication elements and the different options for spot communication that are available for Sun SPOTs.
- Outline:
 - ♦ Introduction
 - ♦ Radiostream communication
 - ♦ Radiogram communication
 - ♦ Broadcasting
 - ♦ Port number usage
 - ♦ HTTP Support

Introduction

- The current version of the Sun SPOT SDK uses the Generic Connection Framework (GCF) to provide radio communication between SPOTs:
 - routed via multiple hops, if necessary,
 - uses a choice of two protocols:
 - Radiostream
 - Radiogram
- The protocols are implemented on top of the MAC layer of the 802.15.4 implementation.

Introduction

- Java packages involved:

- ♦ `import com.sun.spot.io.j2me.radiogram.*;`
- ♦ `import com.sun.spot.io.j2me.radiostream.*;`
- ♦ `import com.sun.spot.peripheral.NoRouteException;`
- ♦ `import com.sun.spot.peripheral.TimeoutException;`
- ♦ `import com.sun.spot.peripheral.radio.*;`
- ♦ `import java.io.DataInputStream;`
- ♦ `import java.io.DataOutputStream;`
- ♦ `import java.io.IOException;`
- ♦ `import javax.microedition.io.Connector;`
- ♦ `import javax.microedition.io.Datagram;`
- ♦ `import javax.microedition.io.DatagramConnection;`

Radiostream communication

Radiostream communication

- The radiostream protocol is a socket-like peer-to-peer protocol.
- Provides reliable, buffered, stream-based communication between two devices.
- To open a connection do:

```
RadiostreamConnection conn = (RadiostreamConnection)  
    Connector.open("radiostream:<destAddr>:<portNo>");
```

where

- `destAddr` is the IEEE Address of the radio at the far end
- `portNo` is a port number in the range 0 to 255 that identifies the particular connection.

Radiostream communication

- To establish a connection both ends must open connections specifying the same `portNo` and corresponding IEEE addresses (`0014.4F01` + Sun SPOT MAC address).
- Once the connection has been opened, each end can obtain streams to send and receive data:

```
DataInputStream dis = conn.openDataInputStream();  
DataOutputStream dos = conn.openDataOutputStream();
```

Radiostream communication

- An example...

Radiostream communication

- An example...

Program 1

Radiostream communication

```
RadiostreamConnection conn =
(RadiostreamConnection)Connector.open("radiostream://0014.4F01.0000.0006:100");

DataInputStream dis = conn.openDataInputStream();
DataOutputStream dos = conn.openDataOutputStream();

try {
    dos.writeUTF("Hello up there");
    dos.flush();
    System.out.println ("Answer was: " + dis.readUTF());

} catch (NoRouteException e) {
    System.out.println ("No route to 0014.4F01.0000.0006");

} finally {
    dis.close();
    dos.close();
    conn.close();
}
```

Radiostream communication

- An example...

Program 2

Radiostream communication

```
RadiostreamConnection conn =
(RadiostreamConnection)Connector.open("radiostream://0014.4F01.0000.0007:100");

DataInputStream dis = conn.openDataInputStream();
DataOutputStream dos = conn.openDataOutputStream();

try {
    String question = dis.readUTF();
    if (question.equals("Hello up there")) dos.writeUTF("Hello down there");
    else dos.writeUTF("What???");
    dos.flush();

} catch (NoRouteException e) {
    System.out.println ("No route to 0014.4F01.0000.0007");

} finally {
    dis.close();
    dos.close();
    conn.close();
}
```

Radiostream communication

- Comments:
 - ♦ Data is sent over the air when the output stream buffer is full or when a `flush()` is issued.
 - ♦ The `NoRouteException` is thrown if no route to destination can be determined.
 - ♦ The stream accesses themselves are fully blocking, i.e., the `dis.readUTF()` call will wait forever (unless a timeout for the connection is specified).
 - ♦ There is no automatic handshaking when opening a stream connection (so if the program 2 is started after the program 1, it may miss the *“Hello up there”* message).

Radiostream communication

- Comments:
 - Every data data transmission between the two devices involves an ACK. The sending stream will always wait until the MAC-level ACK is received from the next hop.

If the next hop is not the final destination, then an ACK is requested from the final destination (asynchronously returned).

If the ACK is not received despite retries, a `NoMeshLayerAckException` is thrown. Then the application has no way of knowing how much data was successfully delivered to the destination.

Radiostream communication

- Comments:
 - The `ChannelBusyException` exception indicates that the radio channel was busy when the SPOT tried to send a radio packet. The normal handling is to catch the exception and retry the send.

It applies to both `radiostream` and `radiogram` protocols.

Radiogram communication

Radiogram communication

- A client-server protocol that provides datagram-based communication between two devices.
- No guarantees about delivery or ordering.
- Datagrams sent over more than one hop could be silently lost, delivered more than once, and be delivered out of sequence.
- Datagrams sent over a single hop will not be silently lost or delivered out of sequence, but they could be delivered more than once.

Radiogram communication

- To open a server connection do:

```
RadiogramConnection conn = (RadiogramConnection)
    Connector.open("radiogram://:<portNo>");
```

where

- `portNo` is a port number in the range 32 to 255 that identifies the particular connection.
- The connection is opened using the default radio channel and default PAN identifier (currently channel 26, PAN 3).

Radiogram communication

- To open a client connection do:

```
RadiogramConnection conn = (RadiogramConnection)
    Connector.open("radiogram://<serveraddr>:<portNo>");
```

where

- `serveraddr` is the IEEE Address of the radio of the server
- `portNo` is a port number in the range 0 to 255 that identifies the particular connection. The port number must match the port number used by the server.

Radiogram communication

- Data is sent between the client and server in datagrams, of type `Datagram`. To get an empty one:

```
Datagram dg =  
    conn.newDatagram(conn.getMaximumLength());
```

- Datagrams support stream-like operations, acting as both a `DataInputStream` and a `DataOutputStream`.
- Datagram size is limited. When using stream operations to read data, the datagram will throw an `EOFException` an attempt is made to read beyond the valid data.

Radiogram communication

- A datagram is sent by asking the connection to send it:

```
conn.send(dg) ;
```

- A datagram is received by asking the connection to fill in one supplied by the application:

```
conn.receive(dg) ;
```

Radiogram communication

- An example...

Radiogram communication

Client end

Radiogram communication

Client end

```
RadiogramConnection conn = (RadiogramConnection)
    Connector.open("radiogram://0014.4F01.0000.0006:100");
Datagram dg = conn.newDatagram(conn.getMaximumLength());

try {
    dg.writeUTF("Hello up there");
    conn.send(dg);
    conn.receive(dg);
    System.out.println ("Received: " + dg.readUTF());
} catch (NoRouteException e) {
    System.out.println ("No route to 0014.4F01.0000.0006");
} finally {
    conn.close();
}
```

Radiogram communication

Server end

Radiogram communication

Server end

```
RadiogramConnection conn =
    (RadiogramConnection)Connector.open("radiogram://:100");
Datagram dg = conn.newDatagram(conn.getMaximumLength());
Datagram dgreply = conn.newDatagram(conn.getMaximumLength());

try {
    conn.receive(dg);
    String question = dg.readUTF();
    dgreply.reset();           // reset stream pointer
    dgreply.setAddress(dg); // copy reply address from input

    if (question.equals("Hello up there")) {
        dgreply.writeUTF("Hello down there");
    } else dgreply.writeUTF("What???");

    conn.send(dgreply);
}
```

Radiogram communication

Server end

```
} catch (NoRouteException e) {  
    System.out.println ("No route to " + dgreply.getAddress());  
  
} finally {  
    conn.close();  
}
```

Radiogram communication

- Comments:
 - Only datagrams obtained from the connection may be passed in send or receive calls on the connection. You cannot obtain a datagram from one connection and use it with another.
 - A connection opened with a specific address can only be used to send to that address. Attempts to send to other addresses will result in an exception.

Radiogram communication

- Comments:
 - It is permitted to open a server connection and a client connection on the same machine using the same port numbers. All incoming datagrams for that port will be routed to the server connection.
 - Currently, closing a server connection also closes any client connections on the same port.

Radiogram communication

- Comments:
 - Each SPOT can open many connections on the same port, as long as they are to different remote SPOTs.
 - Most applications use perhaps one or two ports with fixed numbers.

Radiogram communication

- Comments:
 - However some other applications need to open variable numbers of connections to the same remote SPOT, and for such applications, it is convenient to ask the library to allocate free port numbers as required.

To do this, simply exclude the port number from the url.

```
RadiogramConnection conn = (RadiogramConnection)
    Connector.open("radiogram://0014.4F01.0000.0006");
```

This call will open a connection to the given remote SPOT on the first free port. To discover which port has been allocated, do

```
byte port = conn.getLocalPort();
```

Radiogram communication

- Comments:
 - Server radiogram connections can be opened in a similar way:

```
RadiogramConnection serverConn =  
    (RadiogramConnection)Connector.open("radiogram://");
```

- The same port allocation process works with radiostream connections. However, for radiostream connections, the port number is allocated only after either an input or output stream is opened.

Adjusting connection properties

- The `RadiostreamConnection` and `RadiogramConnection` classes provide a method for setting a timeout on reception.

```
RadiostreamConnection conn = (RadiostreamConnection)
    Connector.open("radiostream://0014.4F01.0000.0006:100");

conn.setTimeout(1000); // wait 1000ms for receive
```

- Notes about using `setTimeout()`:
 - If the caller is blocked waiting for input for longer than the period specified, a `TimeoutException` is generated.
 - `setTimeout(0)` triggers exception immediately if no data.
 - `setTimeout(-1)` turns off timeouts, that is, wait forever.

Broadcasting

Broadcasting

- It is possible to broadcast datagrams. By default, broadcasts are transmitted over 2 hops, so they may be received by devices out of immediate radio range
- Broadcasting is not reliable: datagrams sent might not be delivered. SPOTs ignore the echoes of broadcasts that they transmitted themselves or have already received.
- To perform broadcasting first open a special radiogram connection:

```
DatagramConnection conn = (DatagramConnection)
    Connector.open("radiogram://broadcast:<portnum>");
```

where `<portnum>` is the port number you wish to use.

Broadcasting

- Note that broadcast connections cannot be used to receive. If you want to receive replies to a broadcast then open a server connection, which can be on the same port.
- If you wish broadcasts to travel for more or less than the default 2 hops, do

```
((RadiogramConnection) conn).setMaxBroadcastHops (n) ;
```

where n the required number of hops.

Broadcasting

- Be careful when using basestations:

Imagine a spot sending a radiogram to a basestation which is 2-hops away...

- In dedicated mode, the host application uses the basestation's address and therefore receives it.
- In shared mode, the host application is still one more hop from the host process managing the basestation, and so does not receive the radiogram.
- Similar considerations apply to packets sent from host applications.

Port number usage

Port number usage

- The valid range of port numbers for both the `radiostream` and `radiogram` protocols is 0 to 255. However, for each protocol, ports in the range 0 to 31 are reserved for system use; applications should not use port numbers in this range.

Port number	Protocol	Usage
8	<code>radiogram://</code>	OTA Command Server
9	<code>radiostream://</code>	Debugging
10	<code>radiogram://</code>	http proxy
11	<code>radiogram://</code>	Manufacturing tests
12	<code>radiostream://</code>	Remote printing (Master isolate)
13	<code>radiostream://</code>	Remote printing (Child isolate)
14	<code>radiogram://</code>	Shared basestation discovery
20	<code>radiogram://</code>	Trace route server

HTTP Support

HTTP support

- The HTTP protocol is implemented to allow remote SPOT applications to open HTTP connections to any web service accessible from a correctly configured host computer.
- To open a connection do:

```
HttpConnection connection = (HttpConnection)  
    Connector.open("http://host:[port]/filepath");
```

Where:

- `host` is the Internet host name in the domain notation
- `port` is the port number, which can usually be omitted
- `filepath` is the path/name of the resource being requested from the web server

HTTP support

- The http protocol is implemented as an HTTP client on the Sun SPOT using the socket protocol.

When an http connection is opened, an initial connection is established with the Socket Proxy over radiogram on port 10 (or as specified in the MANIFEST.MF file of your project).

The Socket Proxy then replies with an available radio port that the device connects to in order to start streaming data.

HTTP support

- By default, a broadcast is used in order to find the basestation that will be used to open a connection to the Socket Proxy.

In order to use a specific basestation add the following property to the project's MANIFEST.MF file:

```
com.sun.spot.io.j2me.socket.SocketConnection-  
BaseStationAddress: <IEEE address>
```

- By default, radiogram port 10 is used to perform the initial connection to the Socket Proxy. This can be overridden by adding a property to the project's MANIFEST.MF file.

HTTP support

- An example...

HTTP support

```
HttpConnection connection = (HttpConnection)
    Connector.open("http://www.sunspotworld.com/");
connection.setRequestProperty("Connection", "close");

InputStream in = connection.openInputStream();
StringBuffer buf = new StringBuffer();
int ch;
while ((ch = in.read()) > 0) {
    buf.append((char)ch);
}
System.out.println(buf.toString());

in.close();
connection.close();
```

HTTP support

- In order for the http protocol to have access to the specified URL, the device must be within radio reach of a basestation connected to a host running the Socket Proxy.

This proxy program is responsible for communicating with the server specified in the URL.

The Socket Proxy program is started by

```
ant socket-proxy
```

or

```
ant socket-proxy-gui
```