Mobila applikationer och trådlösa nät

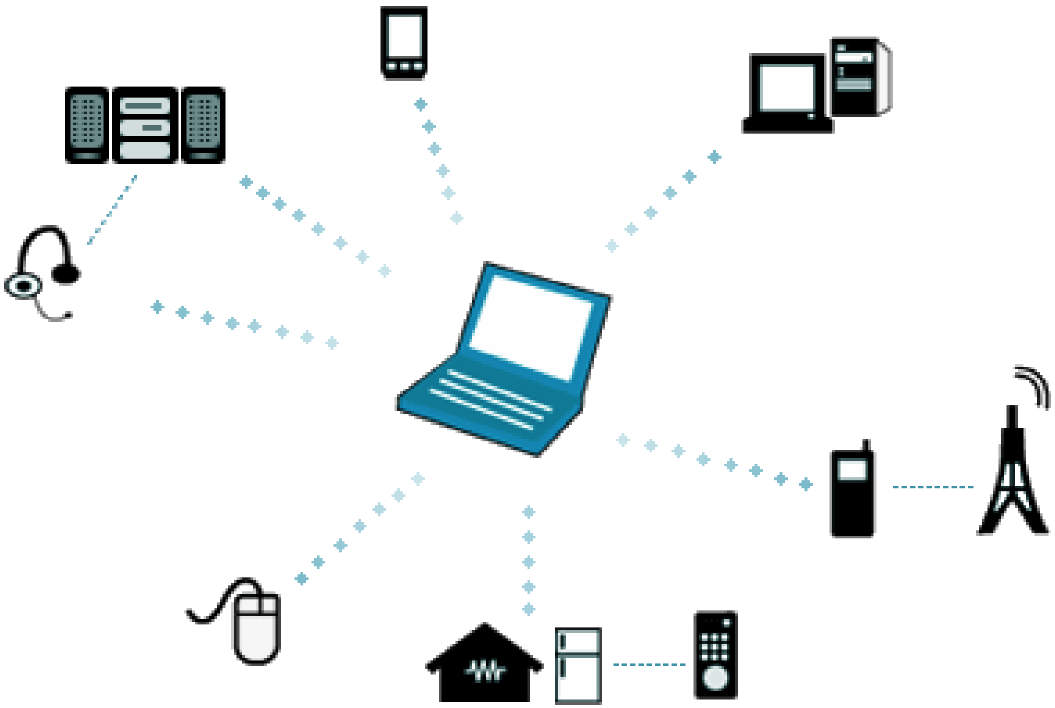
HI1033

Lecturer: Anders Lindström,
anders.lindstrom@sth.kth.se

Lecture 10
Today's topics

- Bluetooth
- NFC

Bluetooth



Bluetooth

- Wireless technology standard for exchanging data over short distances, peer-to-peer
- Ericsson, IBM, Toshiba, Nokia, Intel, ...
- Proprietary open

Bluetooth

- Each device has a Bluetooth chip and antenna
- Radio frequency 2.45 GHz
- Relativly slow; 721 kbit/sec – 3 Mbit/sec (ver. 2.0)

- Frequency jumps, 1600/sec, takes care of interference problems
- Up to 7 connections simultaneously

- Can run in parallell with WiFi (802.11b)

Bluetooth

Class	Min. power, sender	Max. power, sender	Min. range	Typical use
Class 1	0 dBm (1 mW)	20 dBm (100 mW)	<100 m	Devices with no limit on current
Class 2	-7 dBm (0,25 mW)	4 dBm (2,5 mW)	<10 m	Battery powered devices
Class 3	0 dBm (1 mW)	0 dBm (1 mW)	<1 m	Battery powered devices

Bluetooth

- Bluetooth protocols simplify the discovery and setup of services between devices
- Bluetooth devices can advertise all of the services they provide
- A device in discoverable mode on demand transmits
  - Device name
  - Device class
  - List of services
  - Technical information
- Two devices need to be paired to communicate with each other
- Bluetooth v2.1 - Encryption is required for all non Service Discovery Protocol connections

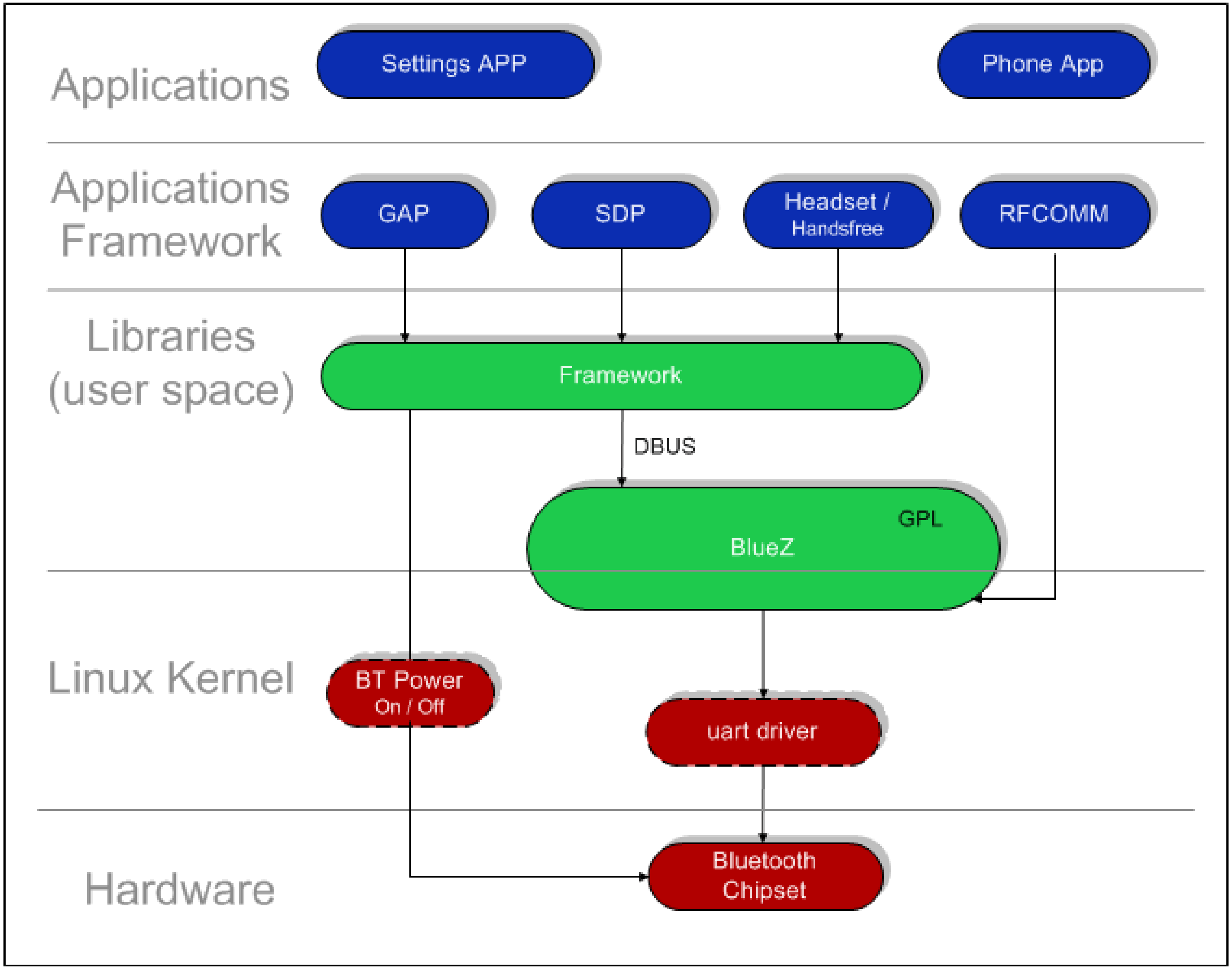
Bluetooth profiles

- Generic Access Profile, Service Discovery Application Profile supported by all devices
- Other, optional, profiles defining e.g.
  - audio/video/image distribution
  - Object exchange (push)
  - Remote control
  - Headsets, hands free
  - SIM access
  - Health Device Profile
  - …

Serial Port Profile (SPP)

- Based on the RFCOMM protocol which provides a simple reliable data stream to the user, similar to TCP
- Emulates a serial cable to provide a simple substitute for existing RS-232
- The basis for other profiles, such as DUN, FAX, HSP and AVRCP

Android Bluetooth stack



Communicate using Bluetooth

Steps:

1. Setting up Bluetooth
2. Finding devices that are either available in the local area (discovery) or already paired
3. Connecting devices
4. Transferring data between devices

Android Bluetooth API

- Supports Bluetooth 2.1
- Using Bluetooth APIs, an Android application can perform the following:
  - Scan for other Bluetooth devices
  - Query the local Bluetooth adapter for paired Bluetooth devices
  - Connect to other devices through service discovery
  - Establish RFCOMM channels
  - Transfer data to and from other devices
  - Manage multiple connections

Android Bluetooth API

- BluetoothAdapter
  - the *local* adapter (Bluetooth radio)
  - the entry point for all interaction
- BluetoothDevice
  - represents a *remote* Bluetooth device
- BluetoothSocket
  - represents the interface for a Bluetooth socket
  - allows an application to exchange data with another device
- BluetoothServerSocket
  - represents an open server socket listening for incoming requests
  - to connect two devices, one must open a server socket
- BluetoothClass
  - describes the general characteristics and capabilities of a Bluetooth device

Uses permissions

- BLUETOOTH
  - required for requesting/accepting a connection and data transfer
- BLUETOOTH_ADMIN
  - required to initiate device discovery and manage bluetooth settings

-
  ```
  <manifest . . .>
     <uses-permission android:name=
       “android.permission.BLUETOOTH” />
  </manifest>
  ```

Set up the local adapter

-
  ```
  BluetoothAdapter adapter =
      BluetoothAdapter.getDefaultAdapter();
  if (adapter != null) {
      // Device does support Bluetooth
  }
  ```

-
  ```
  if(adapter.isEnabled() == false) {
     Intent intent = new Intent(
            BluetoothAdapter.ACTION_REQUEST_ENABLE);
     startActivityForResult(
            intent, REQUEST_ENABLE_BT);
  ```

- The user is prompted to enable the device

Finding devices

- Android devices are not discoverable by default!
- An application can request that the *user* enable discoverability for limited time
- Discover remote devices by
  - Querying for paired devices
  - Starting a device discovery
- Device discovery
  - inquiry scan + page scan
  - > 10 sec, consumes bandwith!

Discovering devices

- startDiscovery() - stopDiscovery() (!)
- Asynchronous
  -Register a BroadcastReceiver to receive information on individual devices being discovered

-
  ```
  IntentFilter filter = new
  IntentFilter(BluetoothDevice.ACTION_FOUND);
  registerReceiver(discoveryReceiver, filter);
  ```

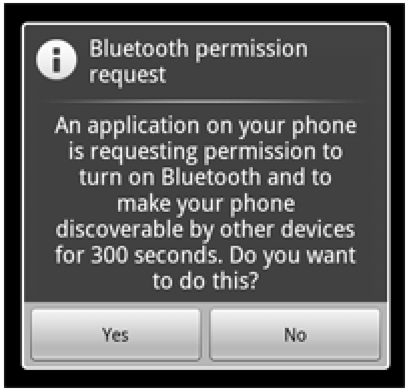
- Don't forget to unregister, e.g. during onDestroy

Discovering devices

```
private class DiscoveryReceiver extends BroadcastReceiver {

    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();

        // Device discovered?
        if (BluetoothDevice.ACTION_FOUND.equals(action)) {

            // Get the discovered device
            BluetoothDevice device =
                      intent.getParcelableExtra(
                             BluetoothDevice.EXTRA_DEVICE);

            // Do something . . .
            arrayAdapter.add(device.getName() + "\n" +
                      device.getAddress());
        }
    }
};
```

Enabling *discoverability*

- Prompts the user
- Makes the local device discoverable to others for 120 (max 300) secs
- Bluetooth is automatically enabled



Bluetooth permission request

An application on your phone is requesting permission to turn on Bluetooth and to make your phone discoverable by other devices for 300 seconds. Do you want to do this?

-
```
Intent intent = new Intent(
    BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
intent.putExtra(
    BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION,
    300);
startActivityForResult(intent, REQ_DISCOVERABLE);
```

Pairing (bonding)

- When a connection is made with a remote device for the first time, a pairing request is automatically presented to the user
- Information about the remote device is stored; device name, class, MAC address, . . .
- A connection can be initiated with a paired device without performing discovery

- Paired vs. Connected:
  - Paired devices are aware of each other's existence, having a shared link-key that can be used for authentication
  - Connected devices currently share an RFCOMM channel

Querying paired devices

```
Set<BluetoothDevice> pairedDevices =
       adapter.getBondedDevices();

if (pairedDevices.size() > 0) {
    for (BluetoothDevice device : pairedDevices) {
        // Show name and MAC address in a ListView
       arrayAdapter.add(device.getName() + "\n" +
                          device.getAddress());
    }
}
```

- Normally:
  First search paired devices, then (if necessary) make a discovery

Connecting devices

- Server listens for incoming connections using BluetoothServerSocket
- Client initiate the connection using a BluetoothSocket + the servers *MAC address*
- Communication via BluetoothSockets and streams
- If not yet paired, user will be prompted for this

- P2P? Prepare each device as a client and as a server, listening for incoming connections?

Server side

- Get a BluetoothServerSocket via
```
listenUsingRfcommWithServiceRecord(
   String name, UUID id)
```

- The UUID identifies your application – must match the client UUID
- Set a time out!
- Call accept() to start listen
  - *blocks* until connection or time out

- Returns a BluetoothSocket for the data transfer
- Close the server socket

Sever side

```
private class AcceptThread extends Thread {

    private BluetoothServerSocket serverSocket;

    public AcceptThread() {
        serverSocket= null;
        try {
            serverSocket =
                  adapter.listenUsingRfcommWithServiceRecord(NAME, MY_UUID);
        } catch (IOException e) { }
    }


    public void run() {
        BluetoothSocket socket = null;
        try {
                socket = serverSocket.accept();
                // Manage the connection, in a separate threa/ASyncTask
                manageConnectedSocket(socket);
         }
        catch (IOException e) { . . . }
        finally {
            serverSocket.close();
        }
    }
```

Server side

```
private class AcceptThread extends Thread {
    private final BluetoothServerSocket serverSocket;

    . . .

    /** Will cancel the listening socket, and cause the
             thread to finish */
    public void cancel() {
        try {
            serverSocket.close();
        } catch (IOException e) { }
    }
}
```

Client side

- Use the *BluetoothDevice* object representing the remote device to get a BluetoothSocket

- `createRfcommSocketToServiceRecord(UUID id)`

- Initiate the connection by calling socket.connect()
  - blocking, call in a separate thread.
- Times out after 12 seconds, throwing an exception
  - close the socket
- If the UUID matches and the remote device accepts the connection, the socket is ready to transfer data

Client side

```
private class ConnectThread extends Thread {
    private BluetoothSocket socket = null;
    private  BluetoothDevice device;

    public ConnectThread(BluetoothDevice device)  {
        this.device = device;
        try {
            socket = device.createRfcommSocketToServiceRecord(MY_UUID);
        } catch (IOException e) { }
    }

    public void run() {
        try {
            socket.connect();
            // Manage the connection in a separate thread/ASyncTask
           manageConnectedSocket(socket);
        }
       catch (IOException connectException) {
            socket.close(); // Unable to connect; close the socket
        }
    }
```

Client side

```
private class ConnectThread extends Thread {
    private final BluetoothSocket socket = null;
    private final BluetoothDevice device;

    . . .

    // Will cancel an in-progress connection,
    // and close the socket
    public void cancel() {
        try {
            mmSocket.close();
        } catch (IOException e) { }
    }
}
```

RFCOMM-socket API

- Prior to Android version 4.x

  ```
  bluetoothSocket = bluetoothDevice.
     createRfcommSocketToServiceRecord(
     STANDARD_SPP_UUID);
  bluetoothSocket.connect();
  ```

- Version 4.x

  ```
  bluetoothSocket = bluetoothDevice.
     createInsecureRfcommSocketToServiceRecord(
     STANDARD_SPP_UUID);
  bluetoothSocket.connect();
  ```

-
  ```
  UUID STANDARD_SPP_UUID = UUID.fromString("00001101-
  0000-1000-8000-00805F9B34FB");
  ```

Transfering data

- getInputStream() / getOutputStream()
- Read and write data to the streams with read(byte[]) and write(byte[]), or
- Add wrapper/filter streams like BufferedReader, PrintWriter, …
- Use a separate thread for all stream reading and writing (read/write calls are blocking)
- *Provide a method to shut down the connection, by raising a flag and closing the socket*

Transferring data

```
private class DataTransferThread extends Thread {
    . . .
    public DataTransferThread(socket) {
        this.socket = socket;
        try {
            sin = socket.getInputStream();
            sout = socket.getOutputStream();
        } catch (IOException e) { }
    }

    public void run() {
        byte[] buffer = new byte[1024];          int bytes;
        // Keep listening to the InputStream until an exception occurs
        while (true) {
            try {
                bytes = sin.read(buffer);
                // Send the obtained bytes to the UI Activity
                handler.obtainMessage(
                          MESSAGE_READ, bytes, -1, buffer).sendToTarget();
            } catch (IOException e) {
                break;
            }
        }
    }
```

Transferring data

```
private class DataTransferThread extends Thread {
    private final BluetoothSocket socket;
    private final InputStream sin;
    private final OutputStream sout;
    . . .

    // Call this from the main Activity to send data
    // to the remote device
    public void write(byte[] bytes) {
        try {
            sout.write(bytes);
        } catch (IOException e) { }
    }

    // Call this from the main Activity to shutdown the connection
    public void cancel() {
        try {
            socket.close();
        } catch (IOException e) { }
    }
}
```

Bluetooth Health Device Profile (HDP)

- BT profile designed to facilitate transmission and reception of Medical Device data
- API available on Android 4.0 (API level 14)

Testing, resources

- Currently, the AVD doesn't support Bluetooth
- Test the BT part of your application on 2 devices, or
- Use a 3rd part simulator https://github.com/cheng81/Android-Bluetooth-Simulator

- Readings
  - Meier – chapter 13
  - http://developer.android.com/guide/topics/wireless/bluetooth.html
  - Bluetooth chat: http://developer.android.com/resources/samples/BluetoothChat/index.html

Near Field Communication

- NFC is a set of short-range wireless technologies, typically < 5 cm
- Radio frequency 13.56 MHz
- Rates ranging from 106 kbit/s to 848 kbit/s
- Very low power consumption
- Initiator and Target - the initiator actively generates an RF field that can power a passive target

Near Field Communication

- Mobile ticketing in public transport, such as Mobile Phone Boarding Pass
- Mobile payment: the device acts as a debit/credit payment card
- Smart poster: the mobile phone is used to read RFID tags
- Bluetooth pairing
- Applications in the future, e.g.
  - Electronic money
  - Identity documents
  - Mobile commerce
  - Electronic keys - car keys, house/office keys, hotel room keys, etc.

Mobile payment

- Primary models for mobile payments:
  - SMS based transactional payments
  - Direct Mobile Billing
  - Mobile web payments (WAP)
  - Contactless Near Field Communication
- NFC: A Mobile phone equipped with a smartcard is brought near a reader module
- No authentication, or authentication using PIN
- Europe, e.g. parking payment

NFC Mobile payment

- Standard by NFC Forum 2004, supported by Nokia and others
- Banks, Payment technology companies and Telecommunications companies has to cooperate...
- Breakthrough 2012? 2013?

NFC and (Smart)phones at present

- Today (Spring 2012)
  - 10% of devices has NFC hardware
  - < 1% are used
- Nokia Money: NFC supported by all new devices, 2011 -
- Apple: NFC chip in Iphone 5? – Nope.
- Android API from version (>) 2.3, android.nfc package (of course, hardware support needed)