Application Development for Mobile and Ubiquitous Computing

6. Energy Challenge

Dr. Christin Groba
Technische Universität Dresden
Chair of Computer Networks
Lecture structure

Application Development

Mobile Business Applications

Cross-Platform Development

Mobile Web Applications
Android
iOS

Mobile Middleware

Location-based Services
Disconnected Operations
Communication Mechanisms
Energy Awareness

Enabling Technologies and Challenges
- Social fitness app
  - Wristband/Smart Watch + Smartphone/Tablet App
  - Server component for data storage and user management

- Functionality
  - Automatic detection and tracking of sports activities
  - Recording of activity states and content (images, videos, track record, curve with pulse, etc.)
  - Activity timeline - Posting own activities and see others' activities in an integrated timeline with text, images, videos, etc.
  - Management of training schedule and planning of training activities
  - Managing competitions with ranking

Motivating example

Dr. Christin Groba
Application Development - 6. Energy Challenge
Location/activity tracking consumes energy
- When should tracking be activated?
- What accuracy is required?
- What sensors should be used?

Communication consumes energy
- When to post activity information?
- Use cellular network or wait for Wifi?

Development
- Is the app consuming more energy than necessary?
- What are common programming pitfalls?
• Energy consumption
  • What to measure?
  • How to measure?
  • Who consumes energy?

• Radio communication
  • WiFi or 3G/LTE?
  • WiFi Direct or Bluetooth?

• Location tracking
  • GPS, WiFi, or GSM?
  • Dynamic provider/frequency selection

• Unexpected energy users
  • Energy bugs
  • Memory leaks
Energy consumption – What to measure?

Remember Watt and Watt-hour?
Energy consumption is a loose term and may describe
a) Energy, i.e. the amount of total energy used over a
   period of time, or
b) Power, i.e. the rate of using energy.

- Battery capacity/energy: 11400 mWh (=3000mAh*3.8V)
- Average power for talking on 3G: 542mW
- Time until phone runs empty: 21h

Power allows for like-for-like comparison irrespective
of how long the actual measurement took.
Energy consumption – How to measure?

- **Electric circuit**
  - Phone connects to external measuring hardware e.g. oscilloscope [1] or Monsoon power monitor
  - Highly accurate but impractical for phones with non-removable battery

![Experimental setup](image1.png)
Energy consumption – How to measure?

- Hardware-internal profiler
  - Profiler reads actual current and voltage from power management chip and converts that into power
  - Accurate for target mobile development platform e.g., Qualcomm Trepn Power for Snapdragon platform [2]
### Power profiles and stats

- Device manufactures publish power profiles i.e., the current $I_c$ a component $c$ draws under a certain voltage.
- Statistics (e.g., with Android battery usage statics) log the time $t_c$, the component $c$ is in use.
- Energy $[\text{mAh}] = \text{sum}(I_c [\text{mA}] \cdot t_c [\text{h}])$ for all active components $c$.
- Stats may not show required level of detail.

---

1 Milliampere hours, technically a unit of charge, is an indicator for energy consumption because the battery voltage during normal operation is typically constant.
Battery life

- Uses API to log remaining battery life as a function of time to compare two apps [3]
- The app that lasts longer, is more energy-efficient

Energy consumption on Android phone [3]
Access battery life in Android [4]

```java
IntentFilter ifilter = new IntentFilter(Intent.ACTION_BATTERY_CHANGED);
Intent batteryStatus = context.registerReceiver(null, ifilter);

int level = batteryStatus.getIntExtra(BatteryManager.EXTRA_LEVEL, -1);
int scale = batteryStatus.getIntExtra(BatteryManager.EXTRA_SCALE, -1);
float batteryPct = level / (float)scale;

Log.d(TAG, "Battery in percent: "+batteryPct);
```

- BatteryManager broadcasts all battery details in a sticky Intent
- No need to register a BroadcastReceiver, because intent is sticky
Energy consumption – Who consumes?

- CPU
  - Clock frequency determines energy consumption
  - Measuring energy consumption of CPU on its own is impractical since CPU switches clock frequency depending on system load
  - Rather part of model-based energy measurements

- Display
  - Energy-intensive if turned on but essential for user interaction
  - If turned on, higher brightness consumes more energy than lower brightness
  - Dimming screen due user inactivity conserves energy
Radio interfaces
- 3G/LTE, WiFi, BT
- Stand-by:
  - Energy for listening for beacons from the base station or access point
  - Beacons may indicate e.g., a hand-over or pending traffic
- Data transfer:
  - Energy for data exchange
  - Size of data and transmission duration matters

Positioning
- GPS, WiFi, GSM, Accelerometer
- Level of detail matters
- Sensing frequency matters
Syncing user account with server: Use 3G/LTE now or wait for WiFi to become available?
- Energy-wise, waiting for WiFi is the better option

- Energy per bit decreases as data size increases [5]

- Throughput does not reach full link capacity due to TCP slow start

- For 10 MB data
  - downlink
    - LTE = WiFi * 1.6
    - 3G = WiFi * 35
  - uplink
    - LTE = WiFi * 2.5
    - 3G = WiFi * 20

![Energy per bit for bulk data transfers](image)
Radio communication – Wifi or 3G/LTE?

- **Weak signal**
  - Network interface compensates weak signal strength by increasing its TX/RX power
  - Repeated network handovers, e.g. during commute, drains battery

- **Developer options**
  - Wait for WiFi to sync with server
  - Queue a set of network requests and process them together
  - Adapt rate of network requests
  - Anticipate the next series of user interactions and pre-fetch required data in bulk
Intermittent Internet access. Transfer data between phones: Use Wifi direct or Bluetooth?
Choice depends on the throughput

For maximum throughput

- BT consumes 2-3 times less energy than WiFi, but
- WiFi transmits more bytes per amount of energy

---

<table>
<thead>
<tr>
<th></th>
<th>Bluetooth</th>
<th>WiFi</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td>RFCOMM</td>
<td>Access point</td>
</tr>
<tr>
<td>sending from phone</td>
<td></td>
<td>TCP</td>
</tr>
<tr>
<td>KB/s</td>
<td>137</td>
<td>1182</td>
</tr>
<tr>
<td>mW</td>
<td>520</td>
<td>1568</td>
</tr>
<tr>
<td>KB/mWs % lost</td>
<td>0.26</td>
<td>0.76</td>
</tr>
</tbody>
</table>

| receiving by phone |           | TCP | UDP | TCP | UDP |
| KB/s          | 128       | 1201 | 627 | 1336 | 618 |
| mW            | 456       | 1504 | 1496 | 1496 | 1448 |
| KB/mWs % lost | 0.28      | 0.80 | 0.42 | 0.89 | 0.43 |

Power consumed (as an offset from a Measured Base) by Bluetooth and WiFi radios when sending and receiving data with iperf [1]
For slow, nearly constant throughput e.g., audio streaming and voice-over-IP

- BT consumes less energy than WiFi
- WiFi consumption increases with number of streamed bits per second
- BT almost does not depend on the bit rate
Radio communication – Wifi Direct or BT?

- **WiFi Direct**
  - More vulnerable to interference (no channel hopping like Bluetooth)
  - Lacks a power-saving mode like WiFi (i.e., when idle requires 20 times more power than WiFi)

- **Bluetooth**
  - Scanning and being discoverable is highly energy-intensive and should be used selectively

- **Developer option**
  - Chose radio communication according to use case
Location-aware app. For repeated location updates use GPS, WiFi, or GSM?
Location tracking: GPS, WiFi, or GSM

- Power consumption for different location providers

- Trade-off accuracy vs. net battery life [6]
  - GPS 10 m vs. 9 h
  - WiFi 40 m vs. 40 h
  - GSM 400 m vs. 60 h

Power consumption on Nokia N95 [6]
Why is GPS so energy-intensive?

- Communication with 3 or more satellites via slow communication channel (50 bits/second) for an extended time
- While the GPS is on, the system cannot enter sleep state to conserve energy
- Mapping software is processing- and energy-intense
Location tracking: GPS, WiFi, or GSM

- **Developer options**
  - Set priority e.g., with Android’s Google Location Services API [7]
    - PRIORITY_HIGH_ACCURACY: Most precise accuracy with mainly GPS-based positioning
    - PRIORITY_BALANCED_POWER_ACCURACY: Accuracy of 100 m with mainly WiFi and cell tower-based positioning
    - PRIORITY_NO_POWER: app does not trigger any location updates, but receives them from other apps
  - Watch out for limitations e.g., in Android 8.0
    - Location for background apps available only a few times each hour to conserve energy
    - Foreground apps like Google Maps not affected
  - Dynamically select location provider and sensing frequency
Example: Alert when in proximity of point-of-interest (POI)

Main idea:
- The closer to POI, the higher the required location accuracy and sensing frequency
- The faster under way, the sooner at the POI, the higher required accuracy and sensing frequency

Determine sensing interval and location provider dynamically based on [8]:
- Distance to point of interest (POI), and
- User’s transportation mode

Increases battery life by 75% due to drastic cut back on GPS [8]
Location-tracking: Dynamic selection

Dynamic location provider and update interval based on [8]

Heading towards the POI, notice:

- Change of location provider
- Selective use of accelerometer, i.e., when actual speed is determined
- Selective use of GPS i.e., when close to POI and driving
- Change of update interval based; pausing when user is idle

Provider: Net, i.e., WiFi or GSM
Location update interval:
- distance/driving speed (regardless of actual speed)

Provider: Net/GPS
Location update interval:
- if Net available: distance/driving speed (regardless of actual speed)
- if Net unavailable: distance/actual speed
  - pause, when idle

Provider: Net/GPS
Location update interval:
- continuous, when driving
  - distance/speed, when walking
  - pause, when idle

Provider: GPS
Location update interval:
- continuous, when moving
  - pause, when idle
Energy bug

an error in the app, OS, hardware, or firmware that causes an unexpected amount of high energy consumption by the system as a whole [9].
No-sleep bug

- OS freezes system after a short period of user inactivity to conserve energy
- Wake lock, provided by API, allows for specifying when a component needs to stay on irrespective of user activity
- App acquires wake lock without ever releasing it

Example: `Partial_WAKE_LOCK` in Android

```java
PowerManager pm = (PowerManager)
    getSystemService(Context.POWER_SERVICE);
PowerManager.WakeLock wl =
    pm.newWakeLock(PowerManager.PARTIAL_WAKE_LOCK, "My Tag");
wl.acquire();
/* screen and keyboard off, CPU stays on during this section */
...
wl.release();
```

If `Partial_WAKE_LOCK` not released, CPU will continue to run even if the user presses the power button.
Unexpected energy users – Energy bugs

- Loop bug
  - App performs periodic but unnecessary task e.g. calendar syncs its own update with itself endlessly
  - Triggered by unexpected external event e.g., remote server crash or change of remote software version

- Immortality bug
  - Buggy app revives after user killed it explicitly and enters same buggy state again continuing to drain energy

- Developer options
  - Release wake locks,
  - Lower wake lock time
  - Analyse power related-events e.g., with Android’s Battery Historian [10]
Unexpected energy users – Energy bugs

Android’s Battery Historian [10]
Steep decline of battery level (black line)

Reason: CPU is running, an app has acquired a wakelock, and the screen is on

Battery historian helps you to understand what events take place, when energy consumption is high
Memory leak
memory that an app occupies although it does no longer need it.
- Typically, users interact with an app only for short duration before they switch to the next.
- Repeated app loading from scratch takes time and energy.
- OS keeps apps that do not currently interact with user in cache [11]
  - Fixed queue size
  - Order least-recently-used (LRU)
- When system runs out of memory, it kills apps in LRU order.
- The more memory an app uses/leaks, the fewer apps can stay in the queue, the more apps need to be reloaded, the more energy is used.
Android: Activity is most leaked object [12]

Leak: asynchronous component is launched and the activity gets destroyed before the component is finished

Example: Handler that posts a message after some specified delay

[9] Distribution of leaked objects in Android programming framework
public class HandlerExample extends AppCompatActivity {

private Handler mLeakyHandler = new Handler();
private TextView myTextBox;

@Override
protected void onCreate(@Nullable Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_samples);
    myTextBox = (TextView) findViewById(R.id.tv_handler);

    // Post a message and delay its execution for 10 seconds.
    mLeakyHandler.postDelayed(new Runnable() {
        @Override
        public void run() {
            myTextBox.setText("Done");
        }
    }, 1000 * 10);
}

@Override
protected void onDestroy() {
    super.onDestroy();
}

Leaking Handler [13]
Anonymous runnable holds a reference of the activity.
If activity is destroyed while message is delayed (here 10 sec), the activity cannot be garbage collected.
public class HandlerExample extends AppCompatActivity {

    private Handler mLeakyHandler = new Handler();
    private TextView myTextBox;

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_samples);
        myTextBox = (TextView) findViewById(R.id.tv_handler);

        // Post a message and delay its execution for 10 seconds.
        mLeakyHandler.postDelayed(new Runnable() {
            @Override
            public void run() {
                myTextBox.setText("Done");
            }
        }, 1000 * 10);
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
        mLeakyHandler.removeCallbacksAndMessages(null);
    }

    Solution A: Non-leaking Handler
    removeCallbacksAndMessages() in activity’s onDestroy() resolves the memory leak
public class HandlerExample extends AppCompatActivity {

    private Handler mLeakyHandler = new Handler();
    private TextView myTextBox;

    @Override
    protected void onCreate(@Nullable Bundle savedInstanceState) {
        super.onCreate(savedInstanceState);
        setContentView(R.layout.activity_samples);
        myTextBox = (TextView) findViewById(R.id.tv_handler);
        // Post a message and delay its execution for 10 seconds.
        mLeakyHandler.postDelayed(new Runnable() {
            @Override
            public void run() {
                myTextBox.setText("Done");
            }
        }, 1000 * 10);
    }

    @Override
    protected void onDestroy() {
        super.onDestroy();
    }
}

Solution B: Non-leaking Handler
Replace with static inner class and weak reference
@Override
protected void onCreate(@Nullable Bundle savedInstanceState) {
    ...
    // Post a message and delay its execution for 10 seconds.
    mLeakyHandler.postDelayed(new MyRunnable(myTextBox), 1000 * 10);
}

private static class MyRunnable implements Runnable {
    WeakReference<TextView> myTextBox;
    public MyRunnable(TextView myTextBox) {
        this.myTextBox = new WeakReference<TextView>(myTextBox);
    }

    @Override
    public void run() {
        TextView mText = myTextBox.get();
        if (mText != null) {
            mText.setText("Done");
        }
    }
}

Static inner class
Weak reference on the activity’s myTextBox
Save the TextView to a local variable because the weak referenced object could become empty at any time
Developer options

- Check for memory leaks e.g., with Android Studio Memory Profiler [14]
- Know the life cycle of an activity e.g., at screen rotation, or when switching between apps
- Follow guidelines for memory-efficient code [15]
While power consumption is important, it needs to be balanced with functionality, user experience, and performance. Be aware of your choices.