Android Developer Fundamentals V2

Alarms and Schedulers

Lesson 8

8.3 Efficient data transfer and JobScheduler

Contents

- Transferring Data Efficiently
- Job Scheduler
  - `JobService`
  - `JobInfo`
  - `JobScheduler`

Transferring Data Efficiently

Transferring data uses resources

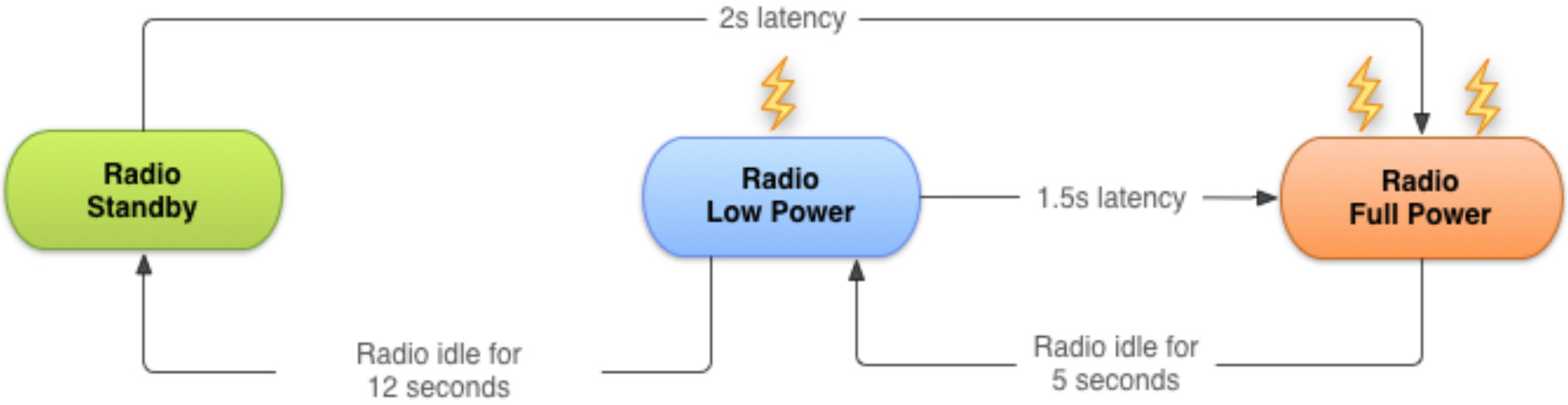
- Wireless radio uses battery.
  - Device runs out of battery.
  - Need to let device charge.

- Transferring data uses up data plans.
  - Costing users real money (for free apps…).

Wireless radio power states

- Full power—Active connection, highest rate data transfer.
- Low power—Intermediate state that uses 50% less power.
- Standby—Minimal energy, no active network connection.

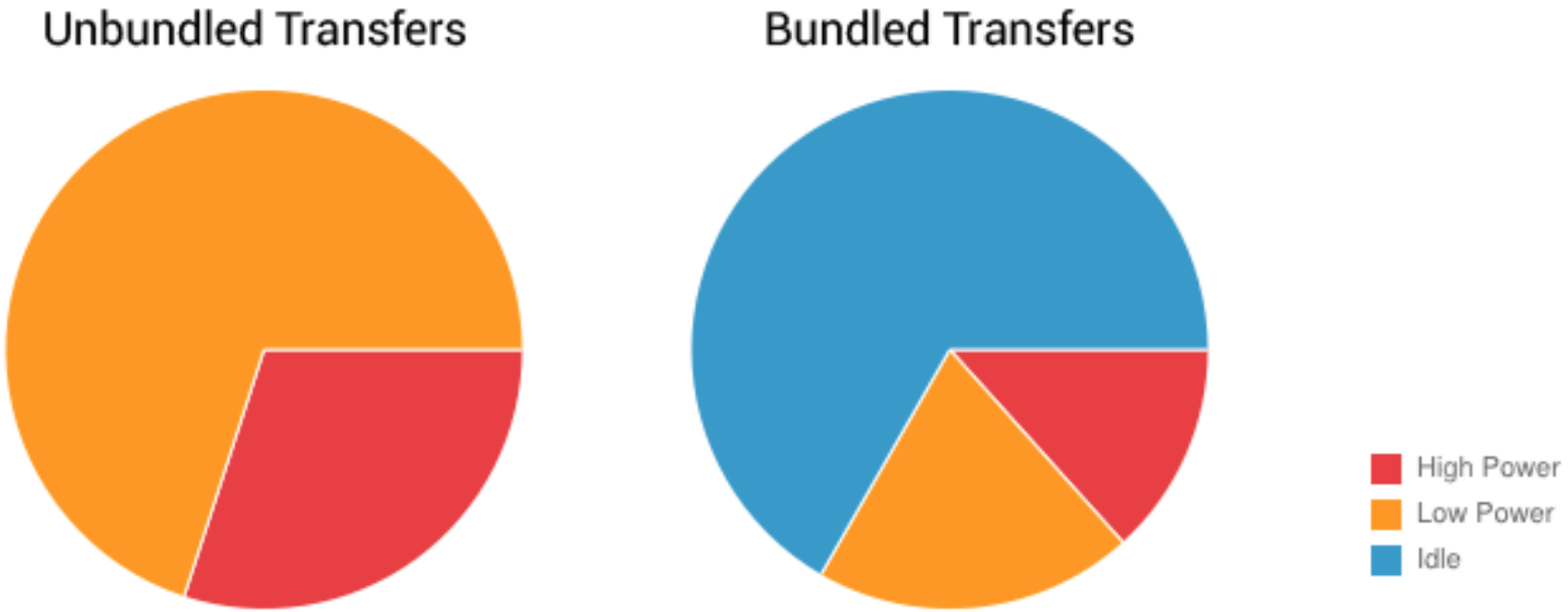
Wireless radio state transitions for 3G



Bundle network transfers

- For a typical 3G device, every data transfer session, the radio draws energy for almost 20 seconds.
- Send data for 1s every 18s—radio mostly on full power.
- Send data in bundles of 3s—radio mostly idle.
- Bundle your data transfers.

Bundled vs. unbundled



Prefetch data

- Download all the data you are likely to need for a given time period in a single burst, over a single connection, at full capacity.
- If you guess right, reduces battery cost and latency.
- If you guess wrong, may use more battery and data bandwidth.

Monitor connectivity state

- Wi-Fi radio uses less battery and has more bandwidth than wireless radio.
- Use `ConnectivityManager` to determine which radio is active and adapt your strategy.

Monitor battery state

- Wait for specific conditions to initiate battery intensive operation.
- `BatteryManager` broadcasts all battery and charging details in a broadcast `Intent`.
- Use a `BroadcastReceiver` registered for battery status actions.

Job Scheduler

What is Job Scheduler

- Used for intelligent scheduling of background tasks.
- Based on conditions, not a time schedule.
- Much more efficient than `AlarmManager`.
- Batches tasks together to minimize battery drain.
- API 21+ (**not in support library**).

Job Scheduler components

- `JobService`—Service class where the task is initiated.
- `JobInfo`—Builder pattern to set the conditions for the task.
- `JobScheduler`—Schedule and cancel tasks, launch service.

JobService

JobService

- `JobService` subclass, implement your task here.
- Override
  - `onStartJob()`
  - `onStopJob()`
- **Runs on the main thread.**

onStartJob()

- Implement work to be done here.
- Called by system when conditions are met.
- Runs on main thread.
- **Off-load heavy work to another thread.**

onStartJob() returns a boolean

`FALSE`—Job finished.

TRUE

- Work has been offloaded.
- Must call **`jobFinished()`** from the worker thread.
- Pass in `JobParams` object from `onStartJob()`.

onStopJob()

- Called if system has determined execution of job must stop.
- … because requirements specified no longer met.
- For example, no longer on Wi-Fi, device not idle anymore.
- Before `jobFinished(JobParameters, boolean)`.
- Return `TRUE` to reschedule.

Basic JobService code

```
public class MyJobService extends JobService {
    private UpdateAppsAsyncTask updateTask = new UpdateAppsAsyncTask();
    @Override
    public boolean onStartJob(JobParameters params) {
        updateTask.execute(params);
        return true; // work has been offloaded
    }
   @Override
    public boolean onStopJob(JobParameters jobParameters) {
       return true;
    }
}
```

Register your JobService

```
<service
    android:name=".NotificationJobService"
    android:permission=
        "android.permission.BIND_JOB_SERVICE"/>
```

JobInfo

JobInfo

- Set conditions of execution.
- `JobInfo.Builder` object.

JobInfo builder object

- Arg 1: Job ID
- Arg 2: Service component
- Arg 3: `JobService` to launch

```
JobInfo.Builder builder = new JobInfo.Builder(
        JOB_ID,
        new ComponentName(getPackageName(),
        NotificationJobService.class.getName()));
```

Setting conditions

```
setRequiredNetworkType(int networkType)
setBackoffCriteria(long initialBackoffMillis, int backoffPolicy)
setMinimumLatency(long minLatencyMillis)
setOverrideDeadline(long maxExecutionDelayMillis)
setPeriodic(long intervalMillis)
setPersisted(boolean isPersisted)
setRequiresCharging(boolean requiresCharging)
setRequiresDeviceIdle(boolean requiresDeviceIdle)
```

setRequiredNetworkType()

`setRequiredNetworkType(int networkType)`

- `NETWORK_TYPE_NONE`—Default, no network required.
- `NETWORK_TYPE_ANY`—Requires network connectivity.
- `NETWORK_TYPE_NOT_ROAMING`—Requires network connectivity that is not roaming.
- `NETWORK_TYPE_UNMETERED`—Requires network connectivity that is unmetered.

setMinimumLatency()

`setMinimumLatency(long minLatencyMillis)`

- Minimum milliseconds to wait before completing task.

setOverrideDeadline()

`setOverrideDeadline(long maxExecutionDelayMillis)`

- Maximum milliseconds to wait before running the task, even if other conditions aren't met.

setPeriodic()

`setPeriodic(long intervalMillis)`

- Repeats task after a certain amount of time.
- Pass in repetition interval.
- Mutually exclusive with minimum latency and override deadline conditions.
- Task is not guaranteed to run in the given period.

setPersisted()

`setPersisted(boolean isPersisted)`

- Sets whether the job is persisted across system reboots.
- Pass in `True` or `False`.
- Requires `RECEIVE_BOOT_COMPLETED` permission.

setRequiresCharging()

`setRequiresCharging(boolean requiresCharging)`

- Whether device must be plugged in.
- Pass in `True` or `False`.
- Defaults to `False`.

setRequiresDeviceIdle()

`setRequiresDeviceIdle(boolean requiresDeviceIdle)`

- Whether device must be in idle mode.
- Idle mode is a loose definition by the system, when device is not in use, and has not been for some time.
- Use for resource-heavy jobs.
- Pass in `True` or `False`. Defaults to `False`.

JobInfo code

```
JobInfo.Builder builder = new JobInfo.Builder(
        JOB_ID, new ComponentName(getPackageName(),
        NotificationJobService.class.getName()))
        .setRequiredNetworkType(JobInfo.NETWORK_TYPE_UNMETERED)
        .setRequiresDeviceIdle(true)
        .setRequiresCharging(true);
JobInfo myJobInfo = builder.build();
```

JobScheduler

Scheduling the job

1. Obtain a `JobScheduler` object form the system.
2. Call `schedule()` on `JobScheduler`, with `JobInfo` object.

```
mScheduler =
    (JobScheduler)getSystemService(JOB_SCHEDULER_SERVICE);
mScheduler.schedule(myJobInfo);
```

Resources

- Transferring Data Without Draining the Battery Guide
- Optimizing Downloads for Efficient Network Access Guide
- Modifying your Download Patterns Based on the Connectivity Type Guide
- `JobScheduler` Reference
- `JobService` Reference
- `JobInfo` Reference
- `JobInfo.Builder` Reference
- `JobParameters` Reference
- Presentation on Scheduling Tasks

What's Next?

- Concept Chapter: 8.3 Efficient data transfer
- Practical: 8.3 Job Scheduler

END