Networking II

Joe Rogers

Android 310

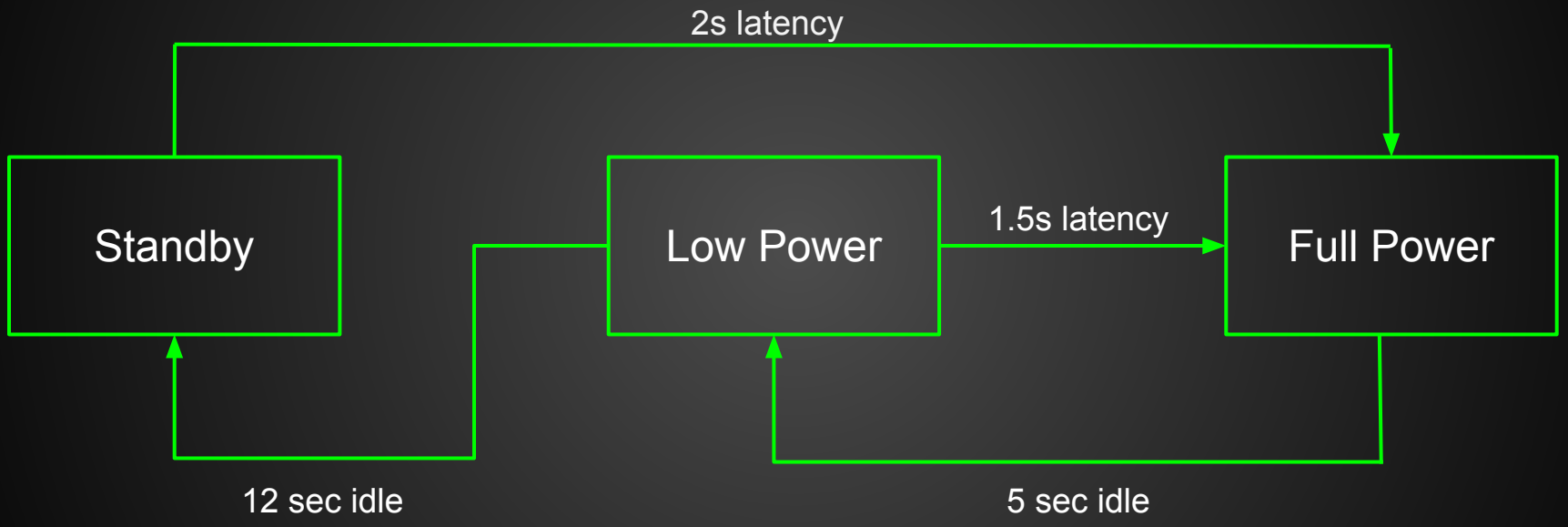
Why worry about networking

- Developers do not always worry about network impacts
  - Tend to develop on wifi and stable environment
  - Thinking of features and not the impact on data usage and battery life.
- However, as Android has progressed, the OS is fingering apps that consume large amounts of data and battery.

First instincts are not always right

- When thinking of network usage, most developer's first instincts are:
  - Fetch data when user opens the app
  - Fetch only what I need for the screen I'm showing.
- Seems reasonable. Developer limiting data use by only fetching data on an as needed basis.

Mobile Radio State Machine



AT&T 3G Network

Ad-hoc fetching

- Lets say app fetches data every 20 second for 1 sec based on user action 3 times. Ie nav to new screen, etc.
- Spends 18 sec at full power. (3 sec of transfer time + 15 sec idle).
- Spends 42 seconds at low power. (36 sec at low power idle + ~5 sec in transition to high power)
- Therefore over the course of a minute the radio was never idle or was there for less that 1 sec, depending on transition timing.

Bundled fetch

- Instead of 3 separate fetches, app fetches all data at once.
- Spends 8 sec at full power. (3 sec + 5 sec full power idle).
- Spends ~14 sec at low power (12 sec idle, plus 2 sec transition to full power)
- Spends ~38 sec idle for the minute.
- This is a notable improvement in battery life for the same amount of data.

Ad-hoc fetching (User perspective)

- Every time user transitions, data is being “fetched” possibly showing a loading indicator.
  - Great on wifi, or high speed lte. Likely very short wait times.
  - Not so good when you get to 3G/2G or a poorly performing network. Those waiting indicators start to go up.

Bundled fetch (User perspective)

- First loads app, may see a loading indicator especially on slow networks.
- Generally for the 2nd/3rd fetches in a bundled fetch, the user should not see a loading indicator as the data is already downloaded. This also improves the user's experience.

Prefetching data

- Essentially front loading data downloads when user starts the app.
- Knowing that a single fetch is going to power the radio for at least 20 seconds, fetching additional information to conserve battery.
- Another advantage is the user not waiting for certain data to download.

Prefetching data cont.

- Based on the radio state machine. Ideally you fetch data that has a 50% chance of being used during the user session.
- Put another way, using the state machine as a guide, in the current user session you could download data for ~6 seconds which would yield up to ~1 to 2MB of data.

Prefetching data cont.

- However, don’t spend 6 seconds and 1-2 MB of downloads to just to fetch data if it is likely to be used less than 50% of time. It is still better to stop earlier.

Real World Example: News app

- Bad App
  - Some news apps fetch only the headlines for the category being shown.
  - Fetch the stories as read and thumbnails the first time they become visible.
  - Results in a lot of random network traffic. Especially if the user is slowly moving up and down the list maybe after reading one story.

Real World Example: News App

- Good app
  - Fetches headlines/thumbs for first category. Then continues to fetch full stories for first category, and headlines/thumbs for other categories as time permits.
  - Even better is app tracks categories user prefers to open and “prefetches” those categories.
  - Alternatively the server tracks stories users tend to open and move those to front of list.

Real World Example: Music app

- While album playing one instinct is to prefetch all of the tracks. However, if user stops after current song, a lot went wasted.
- Better solution is to fetch next track in playlist as user has a 50% chance to listen to it.

The Weather App

- The intent service technique of fetching the current conditions and forecast data in one shot is an example of prefetching data the user may want to see.
  - If the user opens the app likely want to see current conditions 100% of the time.
  - May click to a forecast item 50% of the time especially if its especially good or bad weather.

Batching data

- Another thing apps can do is defer transfers until the next “must” transfer.
  - For example analytic data.
  - Instead of sending data to server as collected, the app should persist the data and then send it the next time it “fetches” data for the user.
  - This may require a little hand-holding with management that wants to see the analytic data ASAP but is much better for battery life.

Background fetching

- Depending on the nature of the app, it may be worth prefetching data in the background in anticipation of a user session.
- However, the risk of the user not seeing the data is much higher so the threshold for performing this should also be higher and less frequent.

Background fetching cont.

- Some ways to increase threshold are:
  - Only fetching if on wifi
  - Only fetching if charging
  - Fetching infrequently based on knowledge of the data or user patterns.

Background fetching example

- Sports app
  - Could fetch upcoming schedule once a week for favorite teams and leagues.
  - If a favorite team is “playing” fetch latest score (or scores for league) every 5-10 min in anticipation of the user checking the score of the league.
  - Or fetch when “notification” of game event arrives anticipating the user opening the app to check progress.

Weather app example

- Fetch current conditions every few hours
  - The “free” API states it may “fail” and to check back in 10 min is not a good experience for the user. Having data 2-4 hours old is better than days old...
- Fetch forecasts once a day.
  - One doesn’t change much and if “fail” to get data likely have a few days in reserve.
- Ideally user chooses to sync/not sync.

Solving the problem

Reduce Data transfer size

- **Problem:** Uncompressed data is significantly more expensive to fetch with respect to radio/latency.
- **Solution:** Ensure server is gzip enabled. HttpUrlConnect requests gzip data by default.
  - Compressed Json/XML is roughly 10x smaller than uncompressed. 50kb vs 500kb.
  - Actually proved this to get server team to enable compression on a project.
  - Will use less battery decompressing vs downloading

Reducing data fetches

- Bundle your fetches
  - As shown, fetching data that will be used 50% of time (up to 6 sec/1-2mb) will be better than than fetching data piecemeal.
- Use caching
  - Use response cache, and/or local cache (database) to reduce data fetches.
  - If data “could” be cached, work with server team to ensure headers are configured to avoid network.

Reducing data fetches cont.

- Know your data
  - If data changes no faster than an hour, ensure you don’t fetch every 5 minutes. If you use a database, query for last fetch time.
  - If background fetches are important, ensure you don’t fetch too frequently or the cost/benefit will be reduced.

Know your network

- Adjust download behavior depending on network type.
  - Download more on wifi
  - Possibly avoid certain data on slower networks (2G).

    For example on news app, avoid thumbnail images on alternate news categories etc.
  - Also if on slow network, avoid any background operations.

Monitor connectivity

- In last weeks sample if no network, the app did a “backoff/retry” solution.
- A better solution is to use broadcast receiver to monitor the CONNECTIVITY_ACTION defined by the connectivity manager. When network is restored, the BR would start the intent service if app still active.

Background Techniques

From best to worst

Use Google Cloud Messaging

- Server sends a GCM message when the data changes. Client not “polling”
  - Requires server and client setup.
  - When client receives message, it schedules a job to download updated data.
  - Example: new email arrives. Gmail server send GCM message to client. Client knows to sync to fetch new messages and notify the user.

Use Job Scheduler

- Job scheduler lets client be woken when certain conditions met such as time, network enabled, charging, etc.
  - Client not woken if condition not met.
  - Client notified if conditions change to stop work
  - OS will “bundle” jobs together. If 3 jobs need network, then OS schedules to use radio together.
  - New in Android 5.0 (but no compatibility)
  - Will persist across boots if desired.

Alarm Manager

- Used as job scheduler prior to 5.0
- Can be used to schedule repeating, exact and inexact events.
- Inexact alarms will not go off precisely every 15 min to allow different alarms to be fire at once.
- No idea if app only one requiring network and network may not be available, etc.
- Must manually reschedule alarm after boot.

Resources

Resources

- Transferring data minimizing battery impact
- Understanding the cell radio (video)
- Job Scheduler
- Connectivity Manager
- Alarm Manager