
Google™
10

Building Push Applications for Android

Debajit Ghosh
May 20, 2010

View live notes and ask questions about this session on Google Wave

http://bit.ly/ac2dmwave

Outline

- Accessing Data in the Cloud
- Polling and Pushing
- Android Cloud to Device Messaging
- Demos
- Summary

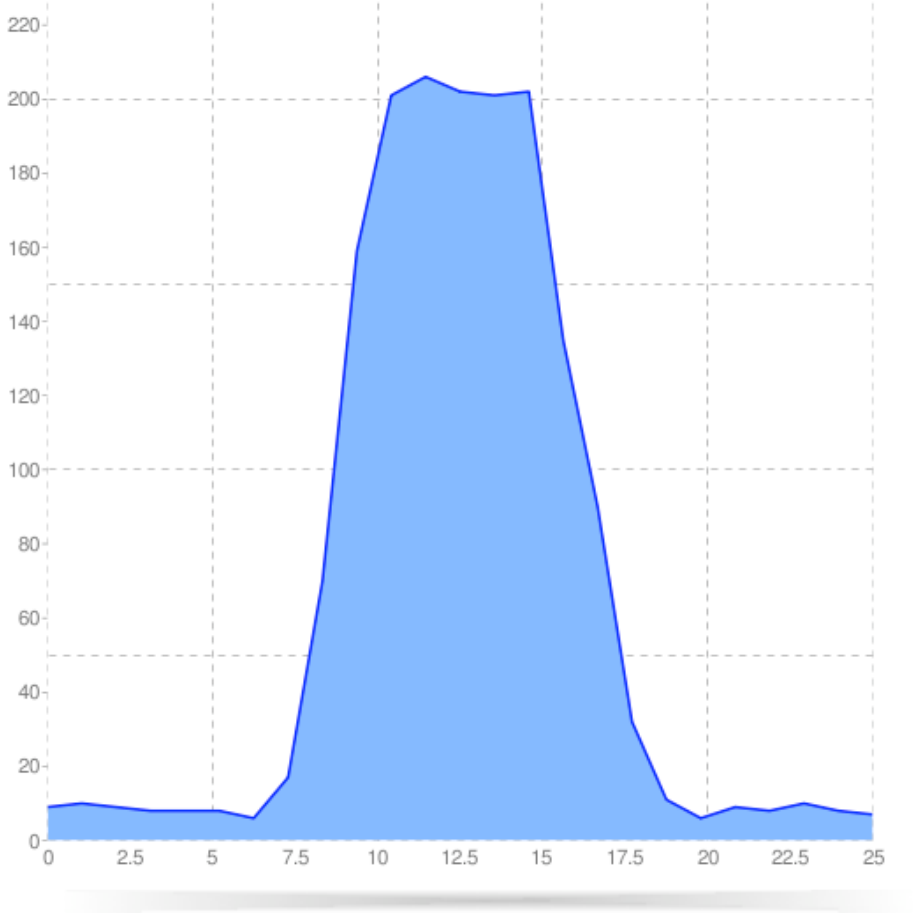
Accessing Data in the Cloud

- Apps provide seamless access to data in the cloud
  - Mobile Alerts
  - Send to Phone
  - Background Sync
- Challenge: How do you keep data on a device fresh?

Polling

- Simple to implement
- Device periodically asks server for new data
  - Radio draws a lot of power, stays on for several seconds
  - Ideally, use If-Modified-Since, If-None-Match, etc.
    - Make no-ops as cheap as possible
- Appropriate for content that changes constantly
  - Stock Quotes, News Headlines
  - Poll infrequently, update on demand

Impact of Polling on Battery



- Baseline: ~5-8 mA
- Network: ~180-200 mA
  - Tx more expensive than Rx
- Radio stays on for few secs
- ~0.50 mAh for a short poll
  - 5m frequency: ~144 mAh / day
  - 15m frequency: ~48 mAh / day

Source: Android development team at Google

When to Poll?

- Tradeoff between freshness and efficiency
  - Poll frequently – more fresh, less efficient
- Desire: *Push*, don't poll
  - Only fetch data when useful

Pushing

- Enables freshness with less impact on battery
  - Only use network when necessary
  - Constant overhead of persistent connection
- Google Contacts, Calendar, Gmail, etc., use push sync
- Can be tricky to implement
- Android Cloud to Device Messaging makes push easy

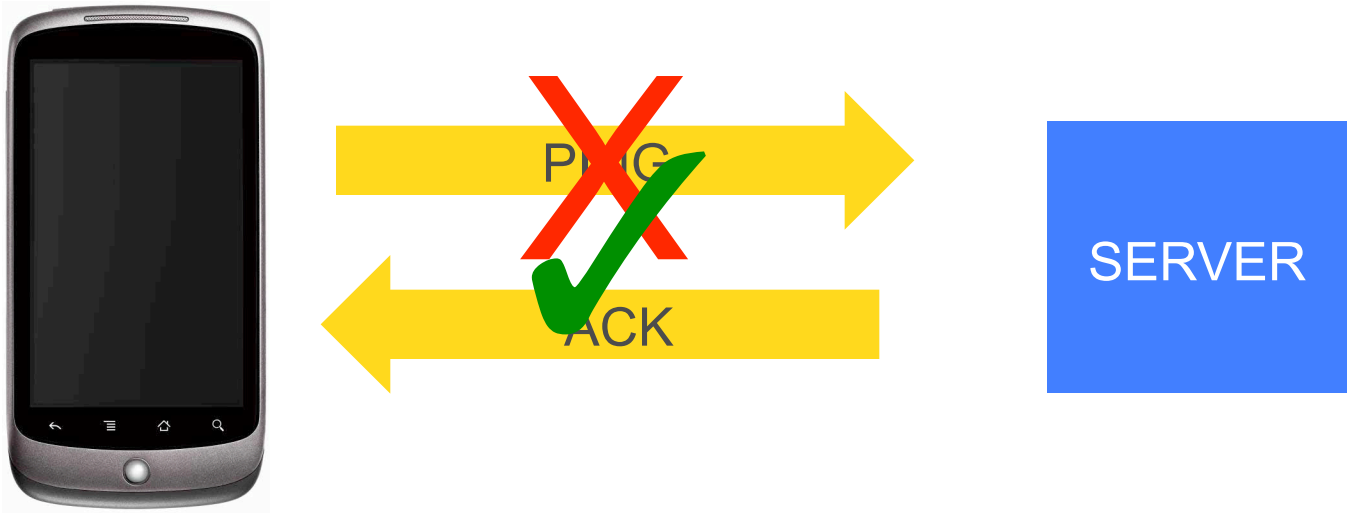
Android Cloud to Device Messaging

- Simple Google API
  - Android 2.2 devices with Market
  - Will be open to all developers
- Uses existing connection for Google services
- Allows servers to send lightweight “data” messages to apps
  - Tell app new data available
  - Intent broadcast wakes up app
  - App supplies UI, e.g., Notification, if/as necessary
- Best effort delivery

Peeking Under the Hood

- Background service
  - Honor background data setting
  - Start when network available
- Maintain connection with server
  - Use heartbeats to keep alive, detect dead connections
- Efficient
  - Minimize per connect overhead
  - Minimize heartbeat frequency
  - Minimize concurrent connections

Heartbeats



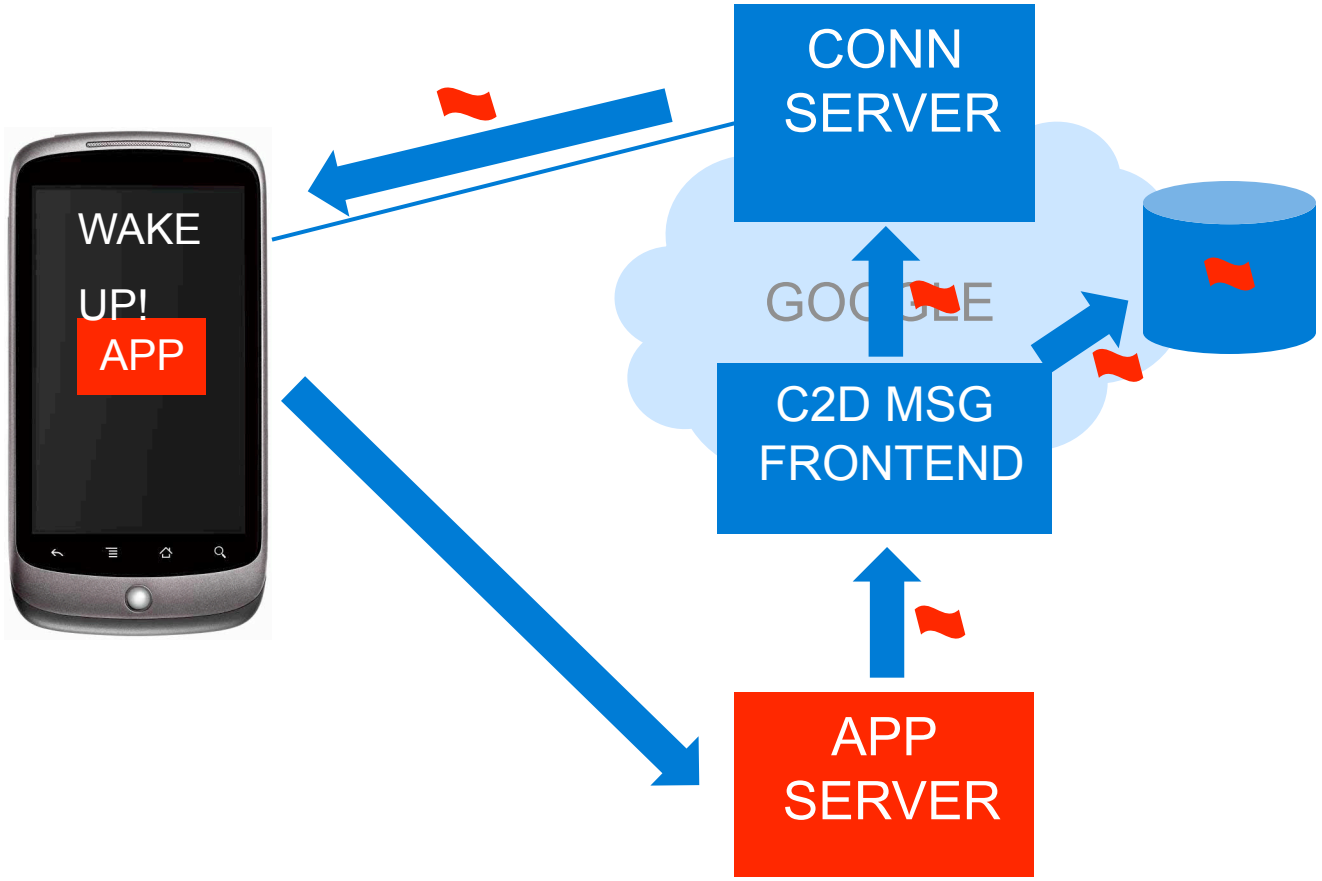
- Use Alarms
  - (Re)schedule pings
  - Wait for acks
- Reconnect when dead

- Can also initiate ping
  - May be half open
- Clean up state when dead

Overview of Lifecycle

- Enabling cloud to device messaging
  - App (on device) registers with Google, gets registration ID
  - App sends registration ID to its App Server
- Per message
  - App Server sends (authenticated) message to Google
  - Google sends message to device
- Disabling cloud to device messaging
  - App can unregister ID, e.g., when user no longer wants push

Life of a Message



Registration – Requesting a Registration ID

```
// Use the Intent API to get a registration ID
// Registration ID is compartmentalized per app/device
Intent regIntent = new
  Intent("com.google.android.c2dm.intent.REGISTER");
// Identify your app
regIntent.putExtra("app",
   PendingIntent.getBroadcast(this, 0, new Intent(), 0);
// Identify role account server will use to send
regIntent.putExtra("sender", emailOfSender);
// Start the registration process
startService(regIntent);
```

Registration – Receiving the Registration ID

- App receives the ID as an Intent
  - com.google.android.c2dm.intent.REGISTRATION
- App should send this ID to its server
- Service may issue new registration ID at any time
  - App will receive REGISTRATION Intent broadcast
  - App must update server with new ID

Registration – Receiving the Registration ID

```
// Registration ID received via an Intent
public void onReceive(Context context, Intent intent) {
  String action = intent.getAction();
  if ("…REGISTRATION".equals(action)) {
    handleRegistration(context, intent);
  } }

private void handleRegistration(Context context, Intent intent){
  String id = intent.getExtra("registration_id");
  if ((intent.getExtra("error") != null) {
    // Registration failed.  Try again later, with backoff.
  } else if (id != null) {
    // Send the registration ID to the app's server.
    // Be sure to do this in a separate thread.
  } }
```

Sending Messages

- Get “ac2dm” auth token, install on server
  - http://code.google.com/apis/accounts/docs/AuthForInstalledApps.html
- Send authenticated POST
  - https://android.apis.google.com/c2dm/send
  - Authorization: GoogleLogin auth=<auth token>
  - URL Encoded parameters
    - registration_id
    - collapse_key
    - (optional) delay_while_idle
    - (optional) data.<key>*

Sending Messages – Response Codes

- 200 OK
  - With “id” – request succeeded, message enqueued
  - With “Error” – request failed
    - QuotaExceeded, DeviceQuotaExceeded: Retry after a while
    - InvalidRegistration, NotRegistered: Stop sending messages
    - MessageTooBig: Reduce size of message
    - MissingCollapseKey: Include collapse key in request
- 401 Not Authorized: Get new auth token
- 503 Service Unavailable: Retry with backoff

Receiving Messages

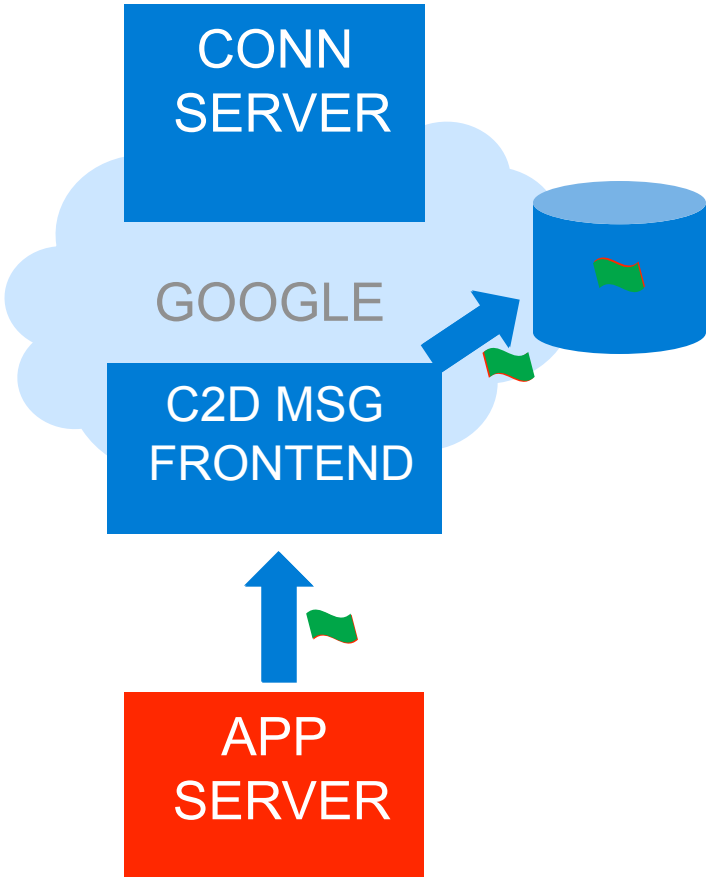
- Device receives message, converts to Intent
- App woken up/started by Intent broadcast
  - com.google.android.c2dm.intent.RECEIVE
  - data.<key>* set as Intent extras
  - App needs com.example.app.permission.C2D_MESSAGE

```
public void onReceive(Context context, Intent intent) {
  String action = intent.getAction();
  if ("…RECEIVE".equals(action)) {
    // Grab a wakelock, use IntentService to do work
  }
}
```

Collapse Keys

- Latest message replaces older ones with same key
- Avoids message explosion for offline device
- App may use multiple collapse keys
  - Correspond to “feed” app will fetch
  - Max of four in flight (per device)
- State should be in app server, not in message
  - Tell app when it should fetch data

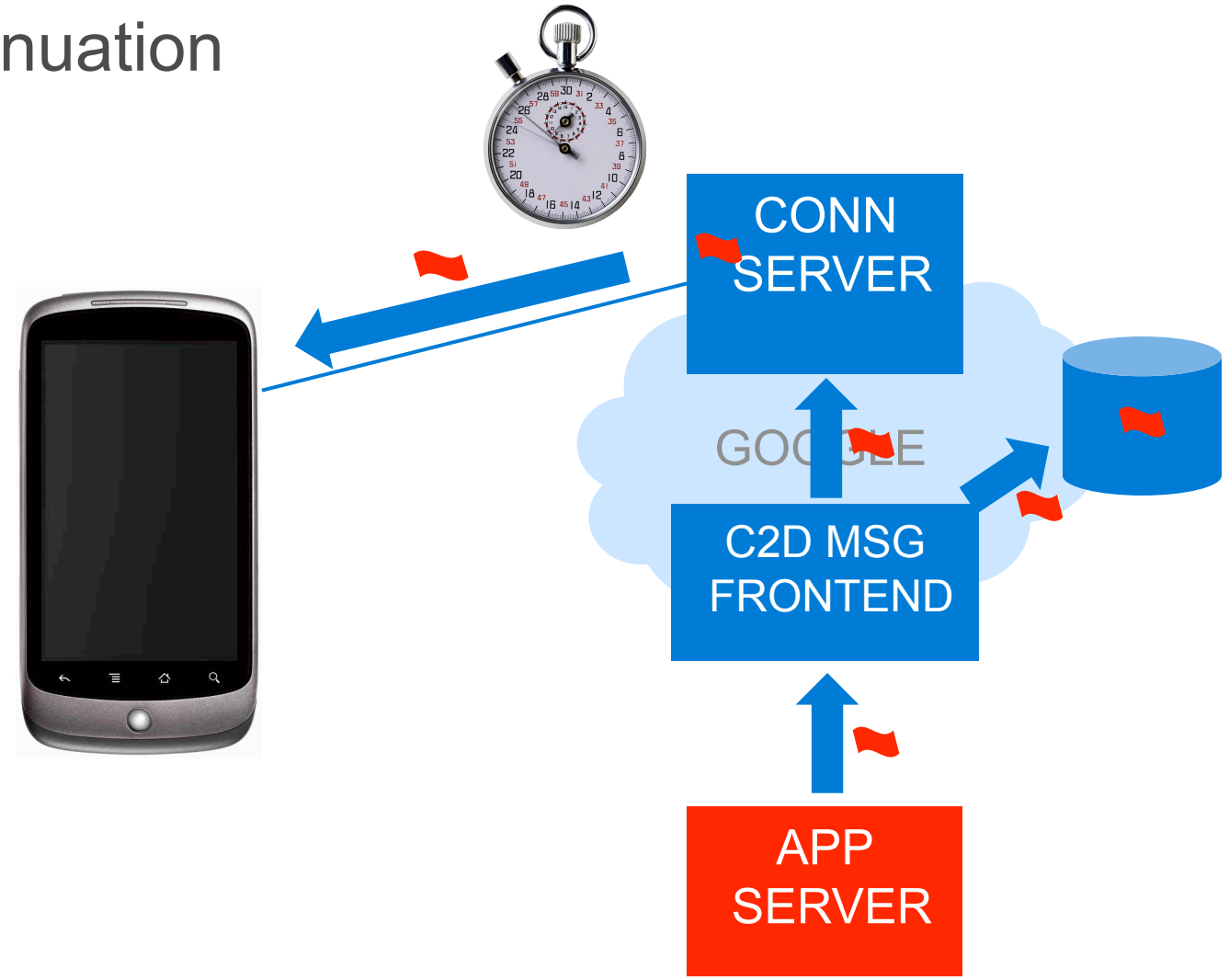
Collapse Keys



Attenuation

- Messages may not be delivered to device immediately
- Protects devices that are receiving many messages
  - Avoid constant radio wakeup
- Attenuation per app/collapse key

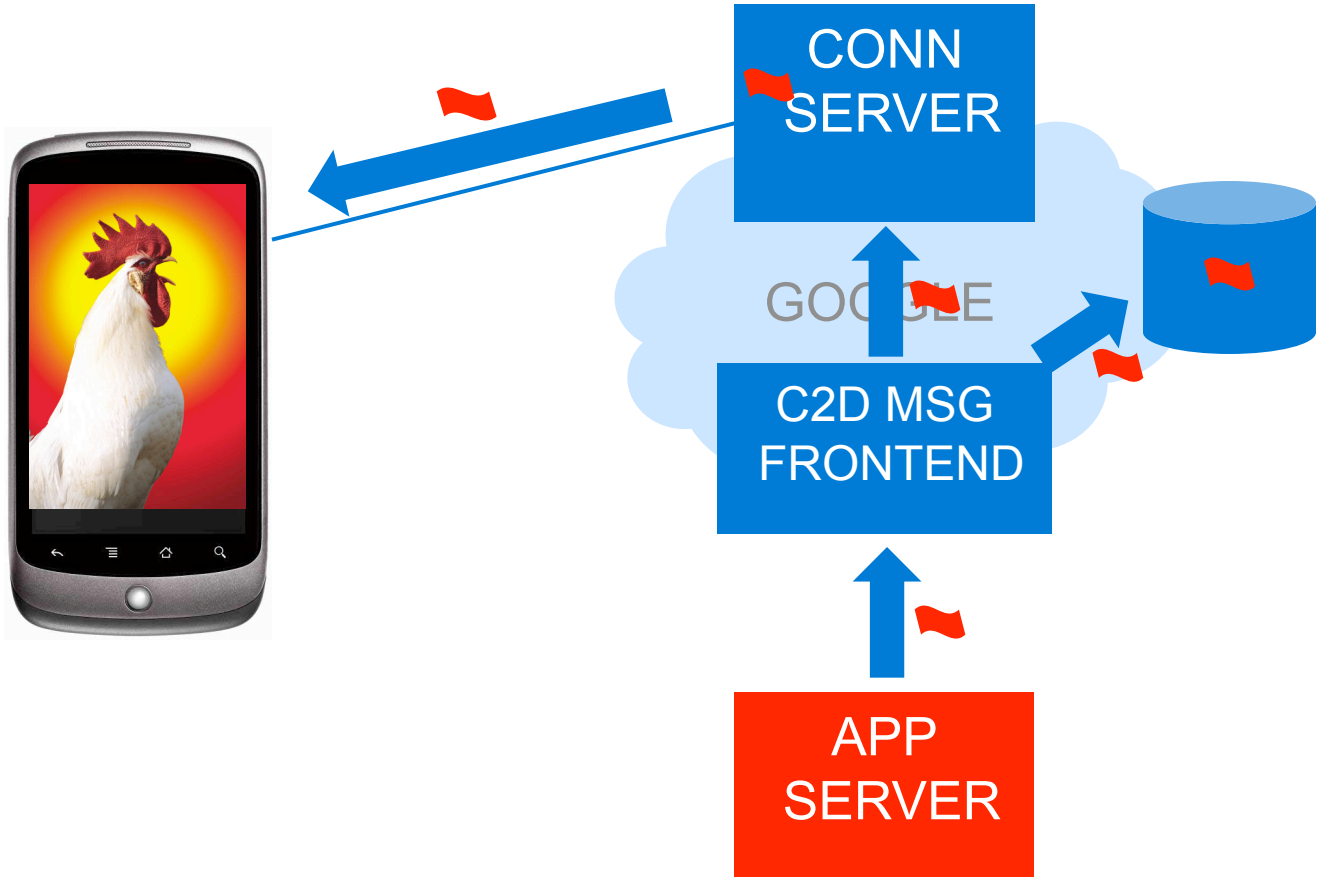
Attenuation



Delay While Idle

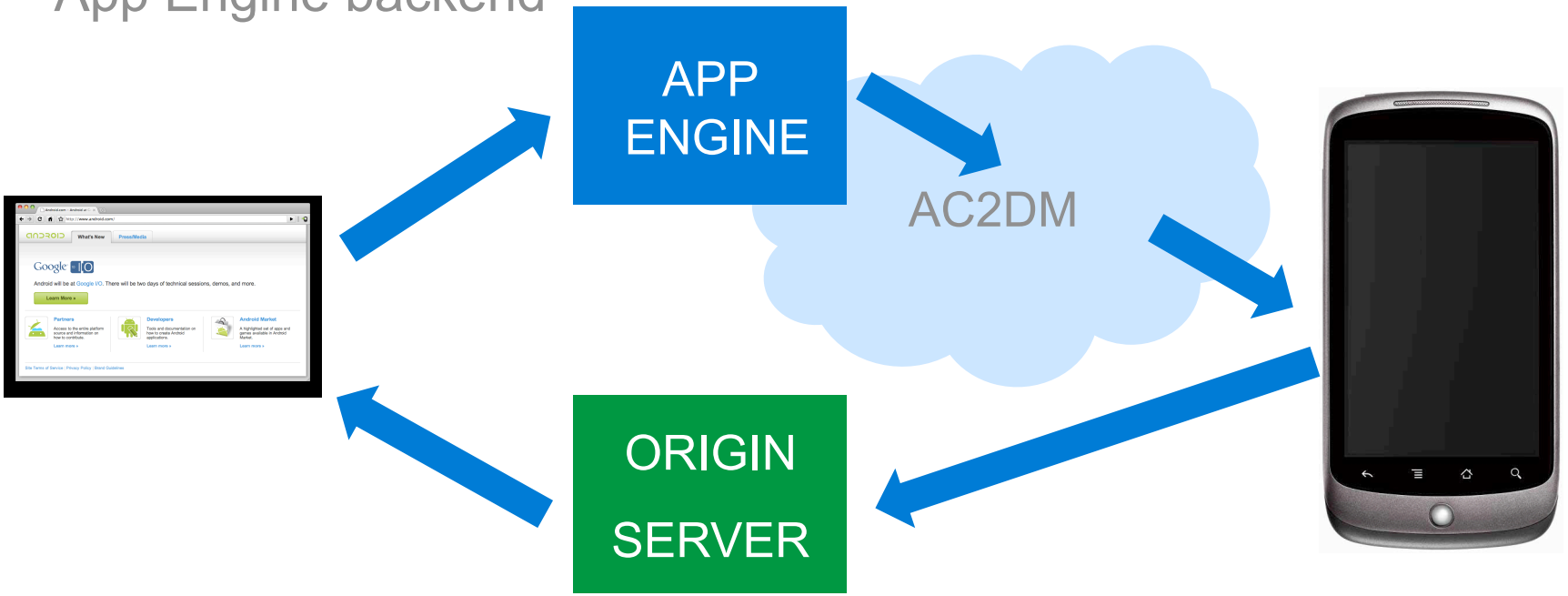
- Device tells Connection Server when screen is on, off
  - Screen off == device is idle
- Apps can request message only be delivered when active
  - Avoid waking up device with info that will not be seen/used
  - e.g., chat presence, friend location updates

Delay While Idle



Demo: Google Chrome to Phone Extension

- Send any web page to Android device
  - Special handling for Maps, YouTube
- Chrome Extension
- App Engine backend



Demo: JumpNote

- Notes, with two way push sync
  - App Engine backend, GWT UI
- Uses Sync Framework
- Uses Android Cloud to Device Messaging
  - Register, Unregister based on auto-sync selection

```
public void onReceive(Context context, Intent intent) {
  String action = intent.getAction();
  if ("…RECEIVE".equals(action)) {
    // Determine account, feed that changed …
    context.getContentResolver.requestSync(account, "…jumpnote", extras);
  }
}
```

Android Cloud to Device Messaging Signup

- Launching in Labs, accepting signups
- Visit http://code.google.com/android/c2dm for details

Summary

- Many Android apps access data in cloud
- Push keeps apps up to date, efficiently
- Android Cloud to Device Messaging makes push simple
- Sign up now
  - http://code.google.com/android/c2dm

View live notes and ask questions about this session on Google Wave

http://bit.ly/ac2dmwave


Google™
10