Web Services

Dominic Duggan
Stevens Institute of Technology

1

WEB SERVICES

2

Representational State Transfer (REST)

- Software architecture for the Web
  - Resources
  - Names (URIs)
  - Representations
  - Uniform Interface (e.g. HTTP)
  - Stateless
  - Hypermedia networks

3

REST Maturity Model

- POX: Plain Old XML
- CRUD: Create-Read-Update-Delete
- HATEOAS: Hypertext as the Engine of Application State

4

REST Verbs

- Retrieve: HTTP GET
- Create:
  - HTTP PUT for new URI or
  - HTTP POST for existing URI (server decides result URI)
- Modify: HTTP PUT to existing URI
- Delete: HTTP DELETE

- Retrieve metadata only: HTTP HEAD
- Check which methods are supported: HTTP OPTIONS
- Merge updates: HTTP PATCH

- No other operations besides these

Example: Amazon Simple Storage Service (S3)

- S3 is based on two concepts
  - Buckets
    - Named container
  - Objects
    - Named piece of data, with metadata
    - Stored in buckets

S3 RPC Interface

- Object-oriented interface to S3
  - `CreateBucket`
  - `ListAllMyBuckets`
- Getter/setter methods on bucket and object "objects"
  - `S3Object.name()`
  - `S3Object.setValue()`
  - `S3Bucket.getObjects()`

7

S3 REST Interface

- Three types of resources
  - List of your buckets

    `https://s3.amazonaws.com`
  - A particular bucket (virtual host)

    `https://`*`name-of-bucket`*`.s3.amazonaws.com`
  - A particular s3 object inside a bucket

    `https://`*`name-of-bucket`*`.s3.amazonaws.com/`*`name-of-object`*

8

S3 REST Interface

- Example:
  - A particular bucket
    `https://jeddak.s3.amazonaws.com`
  - A particular s3 object inside a bucket
    - Object names:
      docs/manual.pdf, docs/security.pdf, talks/snt.pdf
    - Resource URIs:
      `https://jeddak.s3.amazonaws.com/docs/manual.pdf`
      `https://jeddak.s3.amazonaws.com/docs/security.pdf`
      `https://jeddak.s3.amazonaws.com/talks/snt.pdf`

9

S3 REST Interface

- Use HTTP methods as verbs

Verb	**Bucket list**	**Bucket**	**Object**
GET	List buckets	List bucket objects	Get value and metadata
HEAD			Get metadata
PUT		Create bucket	Set object value and metadata
DELETE		Delete bucket	Delete object

10

HTTP

11

HTTP Request

```
GET /index.html HTTP/1.1
Host: www.example.org

...request headers...
```

12

HTTP Response

```
HTTP/1.1 200 OK
Date: Mon, 1 May 2011 21:38:14 GMT
Server: Apache/1.3.34 (Debian) mod ssl/2.8.25
OpenSSL/0.9.8c ...
Last-Modified: Wed, 25 Nov 2009 12:27:01 GMT
ETag: "7496a6-a0c-4b0d2295"
Accept-Ranges: bytes
Content-Length: 2572
Content-Type: text/html
Via: 1.1 www.example.org
Vary: Accept-Encoding
...
```

13

Request Headers

- Accept: for content negotiation
  - Content-Type: response header e.g.
    ATOM (application/atom+xml)
    RDF (application/rdf+xml)
    XHTML (application/xhtml+xml)
    Form-encoded key-value pairs (application/x-www-form-urlencoded)
- Authorization: app-defined auth info
  - WWW-Authenticate: response header with status code of 401 (“Unauthorized”)

14

Request Headers

- Cookie: (non-standard)
  - Save-Cookie: to save cookie on client

15

Response Headers

- Last-modified: time of last modification
  - If-Last-Modified: request header for caching
- Etag: hash of metadata
  - If-None-Match: request header for caching
- Cache-Control: how long to cache
- Upgrade: upgrade protocol e.g. http to https
- Location: URI for newly created resource, redirection, …

16

Response Codes

- 1XX: for negotiation with Web server
  - E.g. 101 (“Switching protocols”) with Upgrade: response header
- 2XX: to signal success
  - E.g. 200 (“Success”), 201 (“Created”), …
- 3XX: redirect clients
  - E.g. 303 (“See other”), 307 (“Temporary redirect”)
- 4XX: client errors
  - E.g. 400 (“Bad request”), 404 (“Not found”), 401 (“Unauthorized”), 403 (“Forbidden”)
- 5XX: server errors
  - E.g. 500 (“Internal server error”)

17

REST CLIENTS

18

Example

Build the search string for a GET

```
private void doSearch
   (String title, String author, String isbn) {
   String encoding = “UTF-8”;
   String searchFeed =
       “http://webservices.amazon.com/onca/xml?”
        + Service=AWSECommerceService"
        + "&SubscriptionId="+WEB_SERVICE_ID”
        + "&Operation=ItemSearch"
        + "&ResponseGroup=Medium“
        + "&SearchIndex=Books";
   String search =
        "&Title="+URLEncoder.encode(title,encoding);
   search += "&Author="+URLEncoder.encode(author,encoding);
   search += "&Keywords="+URLEncoder.encode(isbn,encoding);
```

19

Example

```
URL url = new URL(searchFeed+search);
URLConnection connection = url.openConnection();

HttpURLConnection httpConnection =
    (HttpURLConnection) connection;
int responseCode =
    httpConnection.getResponseCode();
```

Make the HTTP GET request

20

Example

Use DOM to parse response

```
if (responseCode == HttpURLConnection.HTTP_OK) {
   InputStream in = httpConnection.getInputStream();

   DocumentBuilderFactory dbf =
        DocumentBuilderFactory.newInstance();
   DocumentBuilder db = dbf.newDocumentBuilder();
   Document dom = db.parse(in);
   Element docEle = dom.getDocumentElement();
   NodeList nl = docEle.getElementsByTagName("Item");
   if (nl != null && nl.getLength() > 0) {
        for (int i = 0; i < nl.getLength(); i++) {
             Element item = (Element) nl.item(i);
             Element elemTitle =
             (Element) item.getElementsByTagName("Title").item(0);
             ...
        }
   }
}
```

21

Upload Example

```
HttpURLConnection urlConnection =
    (HttpURLConnection) url.openConnection();
try {
  urlConnection.setDoOutput(true);  // Default method is POST
  //urlConnection.setRequestMethod("PUT");
  urlConnection.setChunkedStreamingMode(0);

  OutputStream out =
      new BufferedOutputStream(urlConnection.getOutputStream());
  writeStream(out);

  InputStream in =
      new BufferedInputStream(urlConnection.getInputStream());
  readStream(in);

finally {
  urlConnection.disconnect();
}
```

22

HttpClient

```
import org.apache.commons.httpclient.*;
import org.apache.commons.httpclient.methods.*;
import org.apache.commons.httpclient.params.HttpMethodParams;

// Create an instance of HttpClient.
HttpClient client = new HttpClient();

// Create a method instance.
GetMethod method = new GetMethod(url);

// Provide custom retry handler if necessary
method.getParams().setParameter(HttpMethodParams.RETRY_HANDLER,
               new DefaultHttpMethodRetryHandler(3, false));

// prefer JSON over XML but both are acceptable to the client
method.setRequestHeader("Accept",
               "application/json;q=1.0, application/xml;q=0.8");
```

23

HttpClient

```
try {
  // Execute the method.
  int statusCode = client.executeMethod(method);

  if (statusCode != HttpStatus.SC_OK) {
    System.err.println("Method failed: " + method.getStatusLine());
  }

  // Read the response body.
  byte[] responseBody = method.getResponseBody();

  // Use caution: ensure correct character encoding
  // and is not binary data
  System.out.println(new String(responseBody));

} ...
```

24

HttpClient

```
try {
  ...
} catch (HttpException e) {
  System.err.println("Fatal protocol violation: " +
                      e.getMessage());
  e.printStackTrace();
} catch (IOException e) {
  System.err.println("Fatal transport error: " +
                      e.getMessage());
  e.printStackTrace();
} finally {
  // Release the connection.
  method.releaseConnection();
}
```

25

JSON: JAVASCRIPT OBJECT NOTATION

26

JSON Types

- Basic values: numbers, string, booleans, null
- Arrays:
  ```
  [ “Hello”, “there”, “bud” ]
  ```
- Objects:
  ```
  { “name” : “Joe Smith”,
    “phone” : “201-555-1234” }
  ```

27

```
[
  {"id" : "B0016K40KY",
   "category" : "Faction",
   "title" : "Lawrence of Arabia",
   "pubDate" : "1963-01-30",
   "genres" : [ "Drame", "History" ],
   "info" : { "type" : "film",
              "director" : [ "David Lean" ], ... }
  } ,
  {"id" : "0131934554",
   "category" : "Non-Fiction",
   "title" : "Janson's History of Art",
   "pubDate" : "2006-02-16",
   "genres" : [ "History", "Art" ],
   "info" : { "type" : "book",
              "author" : [ "Penelope Davies" ], ... }
   } ,
   …
]
```

28

JSON in (Android) Java

```
String json = "{"
    + "query": "Pizza",
    + "locations” : [ “07030”, “07040” ]
    + "}";

StringReader srd = new StringReader(json);
JsonReader rd = new JsonReader(srd);

parse(rd);
```

29

JSON in (Android) Java

```
void parse(JsonReader rd) {
   rd.beginObject();
   while (rd.peek() != JsonToken.END_OBJECT) {
      String label = rd.nextName();
      if (“locations”.equals(label)) {
         rd.beginArray();
         while (rd.peek() != JsonToken.END_ARRAY) {
            ...
            zipCodes.add(rd.nextString());
            ...
         }
         rd.endArray();
      }
   }
   rd.endObject();
}
```

30

JSON in (Android) Java

```
void write(JsonWriter wr,
           String query,
           List<String> zips) {
   wr.beginObject();
   wr.name(“query”);
   wr.value(query);
   wr.name(“locations”);
   wr.beginArray();
   for (String zip : zips) {
      wr.value(zip);
   }
   wr.endArray();
   wr.endObject();
}
```

31

REST METHOD

32

Request and Response classes

```
public abstract class Request implements Parcelable {
   // HTTP headers
   public Map<String,String> getHeaders();
   // JSON string for output entity (if any)
   public String getRequestEntity();
   // Parse response headers and input entity (if any)
   public Response getResponse
      (HttpUrlConnection connection,
       JsonReader rd);
}

public abstract class Response implements Parcelable {
   public boolean isValid();
}
```

Initializing Connection

- Check if we are online

```
ConnectivityManager cm = (ConnectivityManager)
   context
      .getSystemService(Context.CONNECTIVITY_SERVICE);

return cm.getActiveNetworkInfo() != null &&
   cm.getActiveNetworkInfo().isConnectedOrConnecting();
```

- Construct the connection

```
connection = (HttpURLConnection) url.openConnection();
```

Connection Properties

```
connection.setRequestProperty("USER_AGENT", ...);
connection.setRequestMethod(method);
connection.setUseCaches(false);
connection.setRequestProperty("CONNECTION", "Keep-Alive");

connection.setConnectTimeout(...);
connection.setReadTimeout(...);

/*
 * App-specific headers
 */
Map<String,String> headers = request.getHeaders();
for (Entry<String,String> header : headers.entrySet()) {
   connection.addRequestProperty(header.getKey(),
                                 header.getValue());
}
```

35

Output Request Entity

```
void outputRequestEntity(Request request) throws IOException {
   String requestEntity = request.getRequestEntity();
   if (requestEntity != null) {
      connection.setDoOutput(true);
      connection.setRequestProperty("CONTENT_TYPE",
                                    "application/json");
      byte[] outputEntity = requestEntity.toBytes("UTF-8");

      connection
          .setFixedLengthStreamingMode(outputEntity.length);
      OutputStream out = new BufferedOutputStream(
                              connection.getOutputStream());
      out.write(outputEntity);
      out.flush();
      out.close();
   }
}
```

36

Error Checking

```
void throwErrors (HttpURLConnection connection)
      throws IOException {
   final int status = connection.getResponseCode();
   if (status < 200 || status >= 300) {
      String exceptionMessage = "Error response "
        + status + " "
        + connection.getResponseMessage()
        + " for " + connection.getURL();
      throw new IOException(exceptionMessage);
   }
}
```

37

Execute Request

```
outputRequestEntity(request);
connection.setDoInput(true);
connection.connect();
throwErrors(connection);

JsonReader rd = new JsonReader(
                 new BufferedReader(
                  new InputStreamReader(
                   connection.getInputStream())));

Response response = request.getResponse(connection, rd);
rd.close();
if (response.isValid()) {
   return response;
}
```

38

Execute Streaming Request

```
connection.setDoOutput(true);
connection.setChunkedStreamingMode(0);
connection.setDoInput(true);

connection.connect();
throwErrors(connection);

return new StreamingResponse(
              connection,
              request.getResponse(connection));

public static class StreamingResponse {
   public HttpURLConnection connection;
   public Response response;
}
```

39

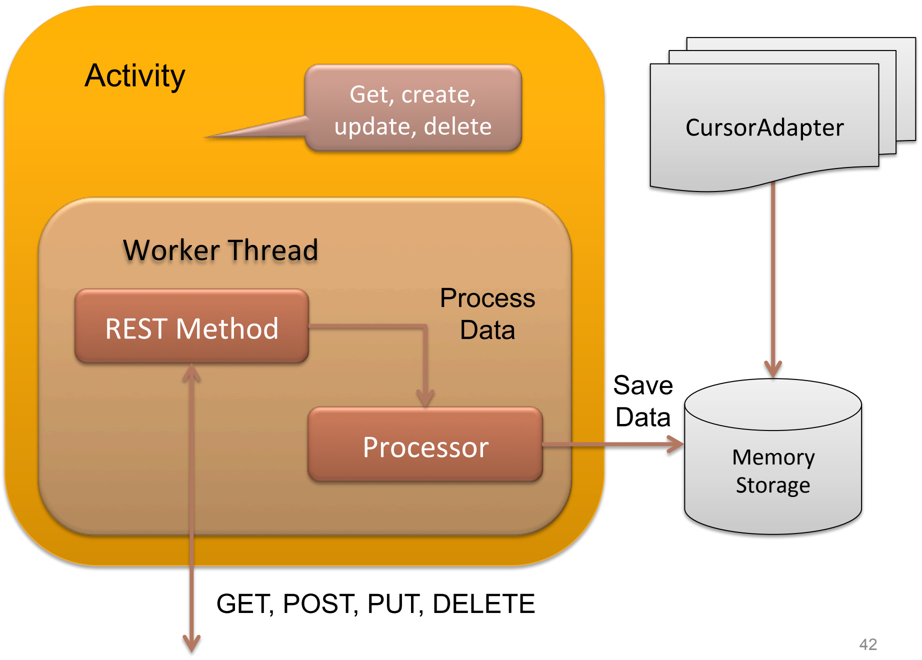
Process Streaming Request

```
StreamingResponse sr = perform(request, ...);
Connection = sr.connection;
Response = sr.response;

// If we assume JSON data (could be e.g. multipart)
OutputStream os = connection.getOutputStream();
JsonWriter wr = new JsonWriter(
                  new BufferedWriter(
                     new OutputStreamWriter(os, “UTF-8”)));
JsonReader rd = ...

// Write streaming output data, read streaming input data
wr.flush();
wr.close();
rd.close();
connection.disconnect();
```

40

Based on slides by Virgil Dobjanschi

REST CLIENT ARCHITECTURE FOR ANDROID



Problems

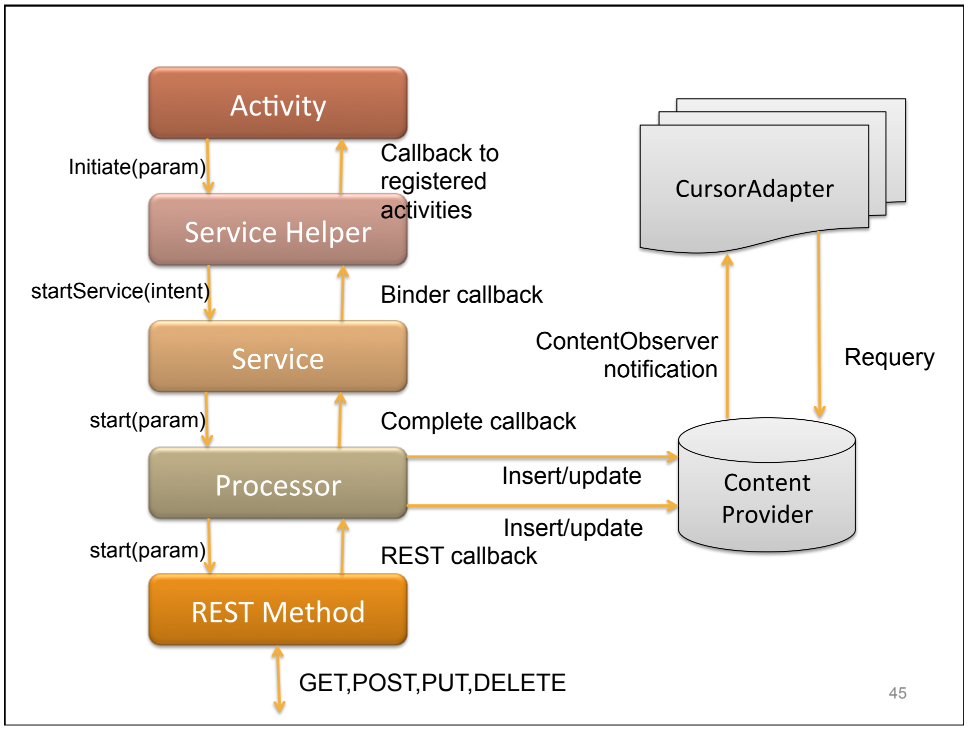
- OS may shut down the process
- Data is not persistently stored

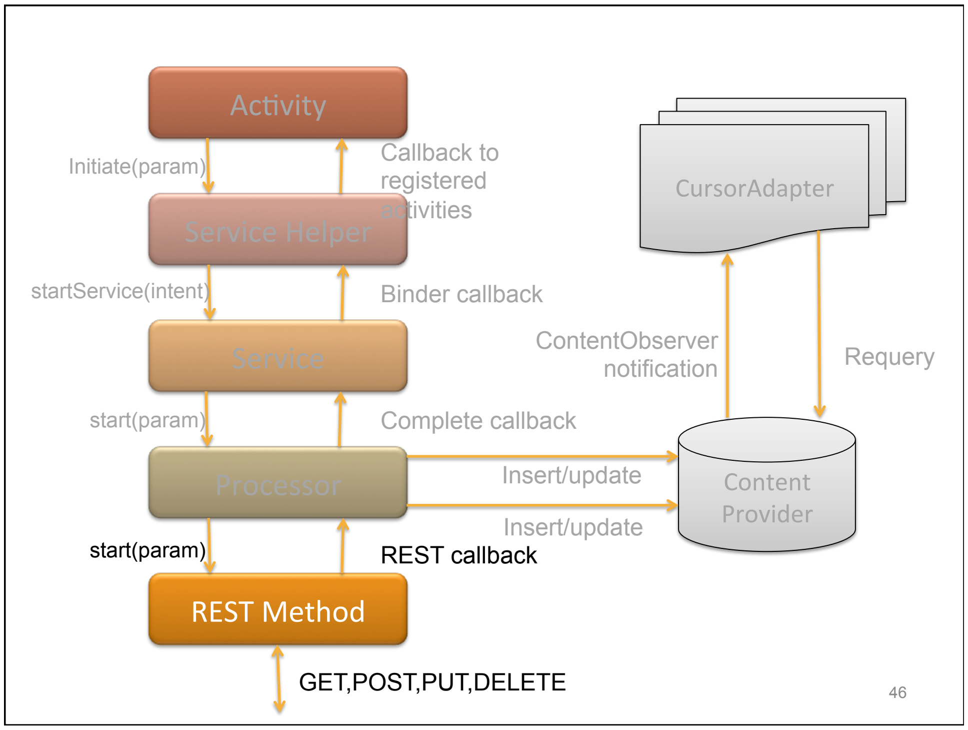
43

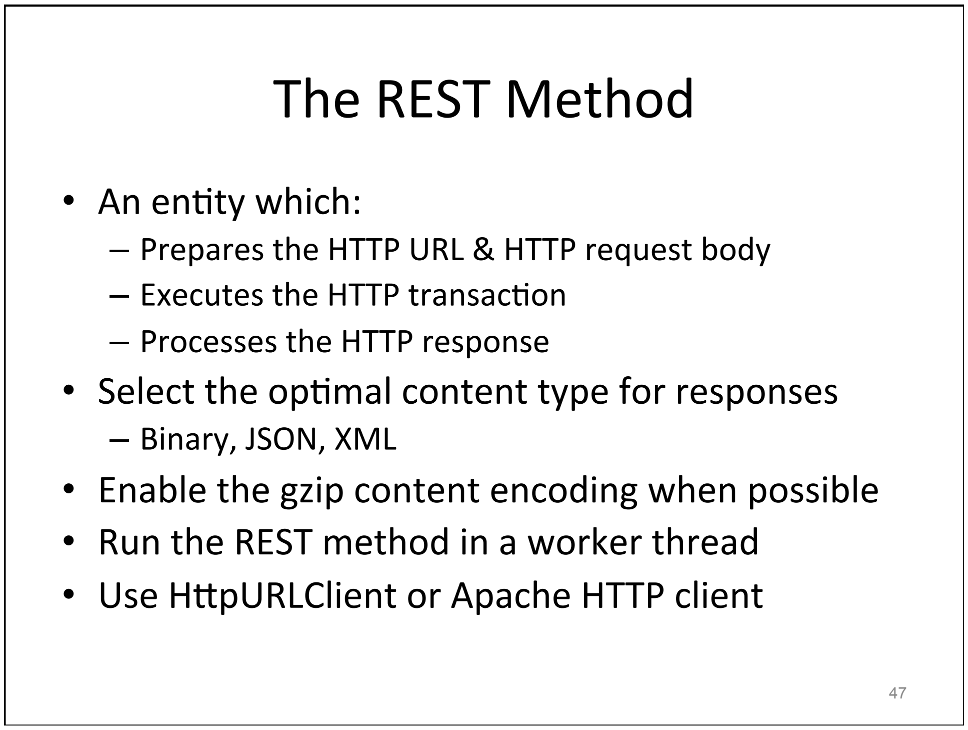
REST Design Patterns

- **Data-Driven: Service API**
- Data-Driven: ContentProvider API
- Control-Driven: Service API

44


Activity
Initiate(param)
Callback to registered activities
Service Helper
startService(intent)
Binder callback
Service
start(param)
Complete callback
Processor
start(param)
REST callback
REST Method
GET,POST,PUT,DELETE
CursorAdapter
ContentObserver notification
Requery
Content Provider
Insert/update
Insert/update
45

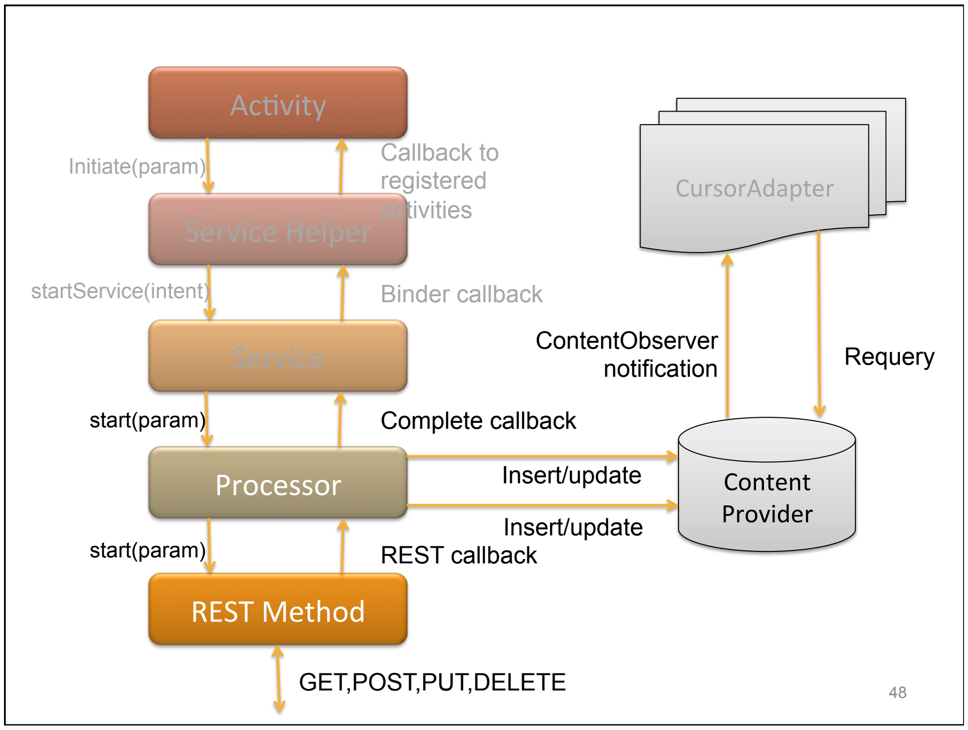

Activity
Initiate(param)
Callback to registered activities
Service Helper
startService(intent)
Binder callback
Service
start(param)
Complete callback
Processor
start(param)
REST callback
REST Method
GET,POST,PUT,DELETE
CursorAdapter
ContentObserver notification
Requery
Content Provider
Insert/update
Insert/update
46



The REST Method

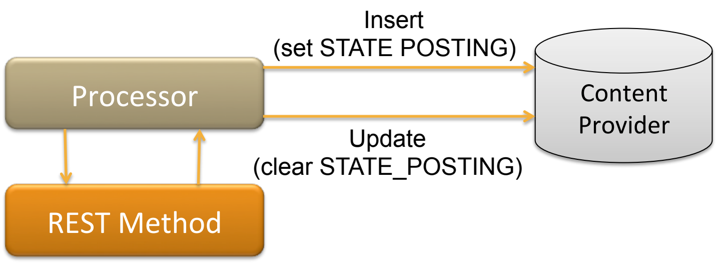
- An entity which:
  - Prepares the HTTP URL & HTTP request body
  - Executes the HTTP transaction
  - Processes the HTTP response
- Select the optimal content type for responses
  - Binary, JSON, XML
- Enable the gzip content encoding when possible
- Run the REST method in a worker thread
- Use HttpURLClient or Apache HTTP client

47

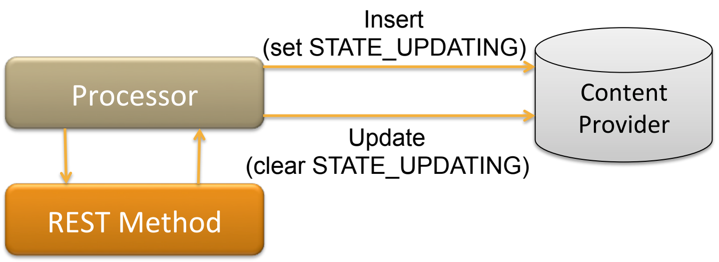


48

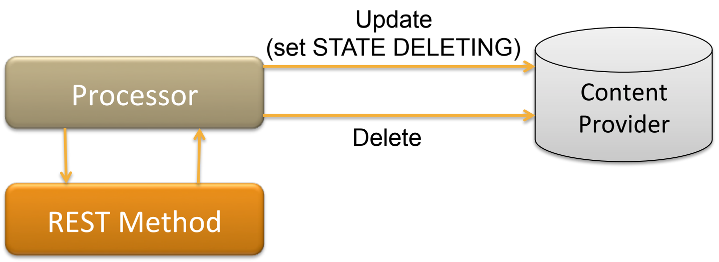
POST



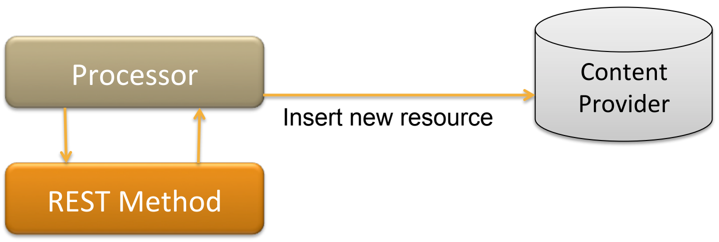
PUT

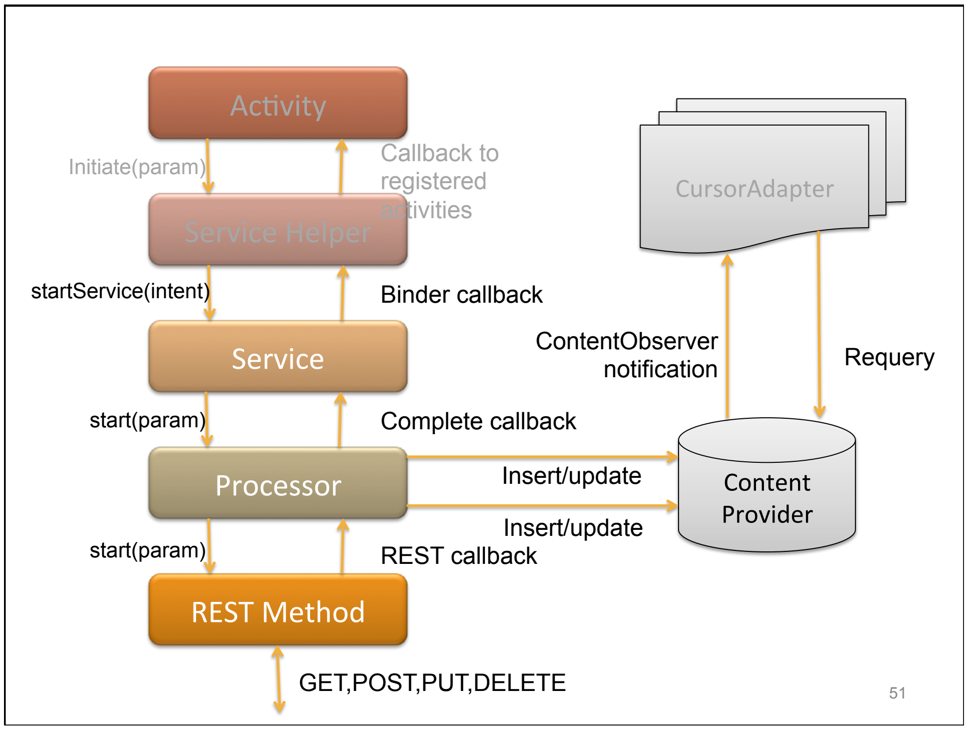


DELETE



GET

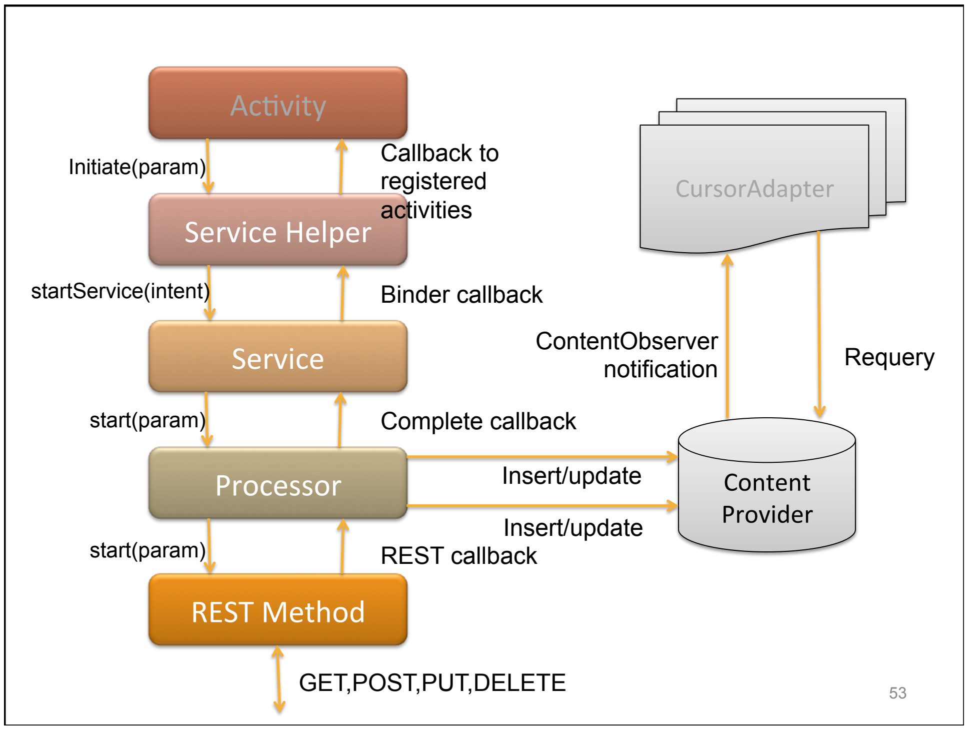


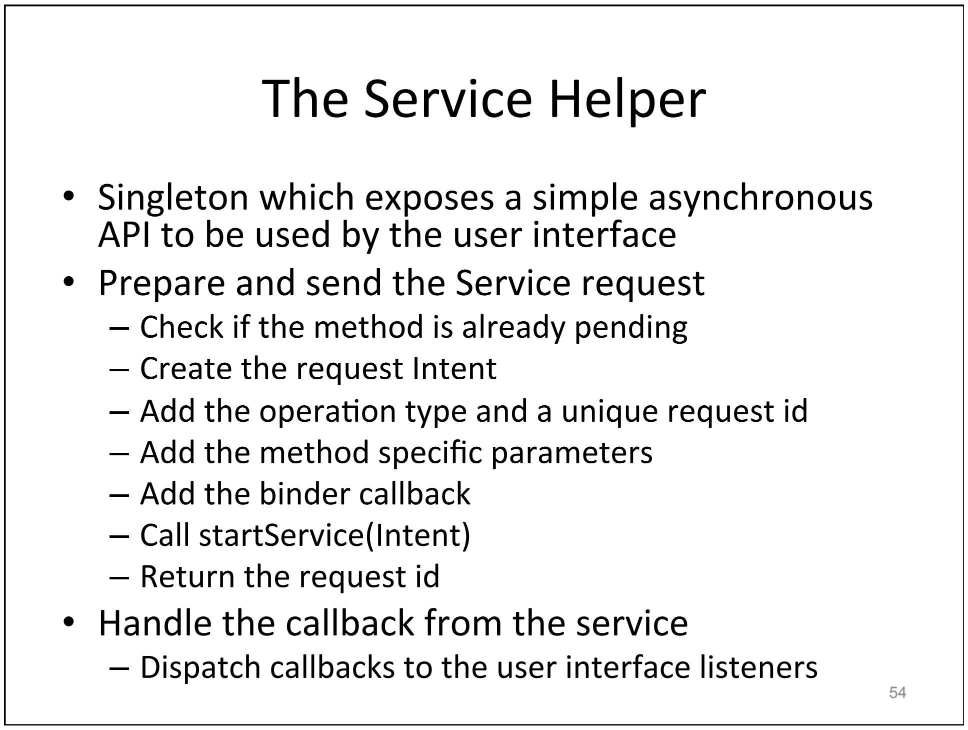


The Service

- The role of the service
- Forward path
  - Receives the Intent sent by the Service Helper and starts the corresponding REST Method
- Return path
  - Handles the Processor callback and invokes the Service Helper binder callback
- Implement a queue of downloads

52

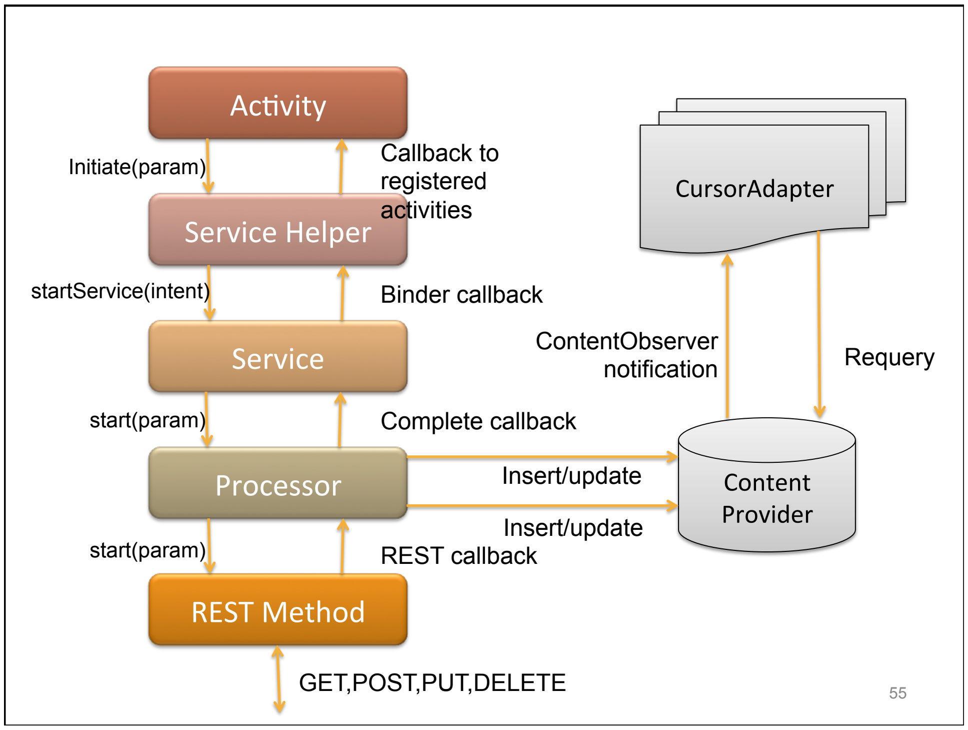




The Service Helper

- Singleton which exposes a simple asynchronous API to be used by the user interface
- Prepare and send the Service request
  - Check if the method is already pending
  - Create the request Intent
  - Add the operation type and a unique request id
  - Add the method specific parameters
  - Add the binder callback
  - Call startService(Intent)
  - Return the request id
- Handle the callback from the service
  - Dispatch callbacks to the user interface listeners

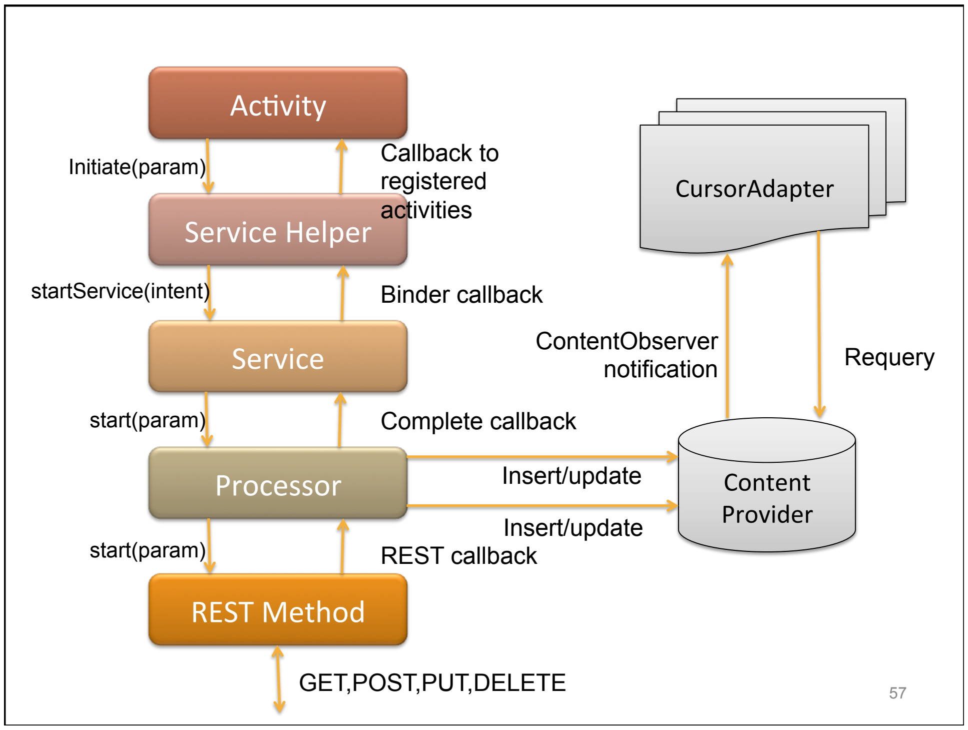
54



Handling REST Method in an Activity

- Add an operation listener in onResume and remove it in onPause
- Consider these cases for activity:
  – Still *active* when the request completes
  – *Paused then resumed* and then the request completes
  – *Paused when the request completes* and then Activity is resumed
- CursorAdapter handles the ContentProvider notification by implementing a ContentObserver

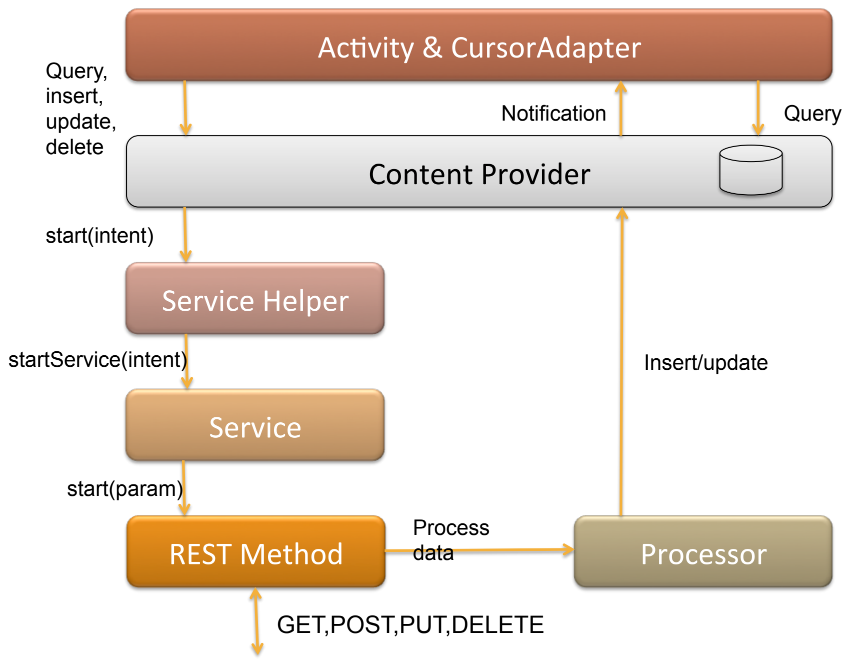
56



REST Design Patterns

- Data-Driven: Service API
- **Data-Driven: ContentProvider API**
- Control-Driven: Service API

58



REST Design Patterns

- Data-Driven: Service API
- Data-Driven: ContentProvider API
- **Control-Driven: Service API**

Conclusions

- Do not implement REST methods inside Activities
- Start long running operations from a Service
- Persist early & persist often
- Minimize the network usage
- Use a sync adapter to execute background operations which are not time critical
  - Google Cloud Messaging (GCM)

61

SHARING DATA THROUGH THE CLOUD: GCM

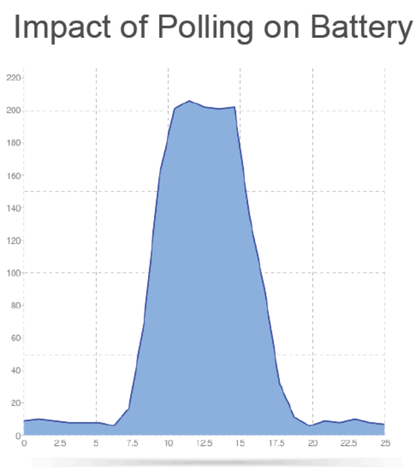
62

Accessing Data in the Cloud: Polling

- Simple
  - Power issues
  - 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

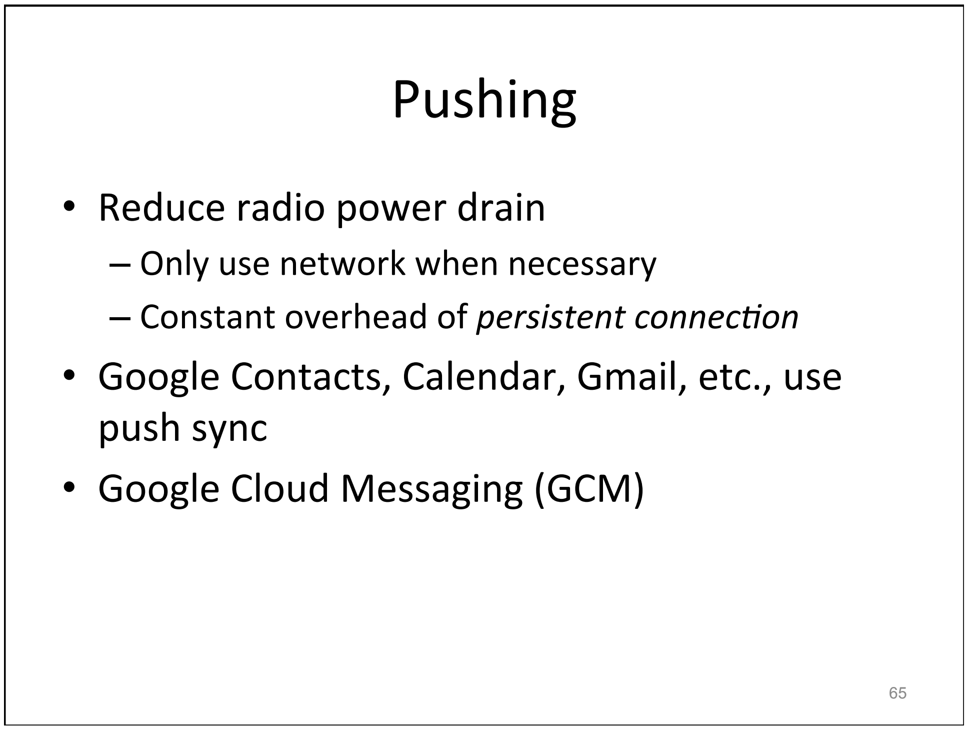
63

Cost of Polling



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

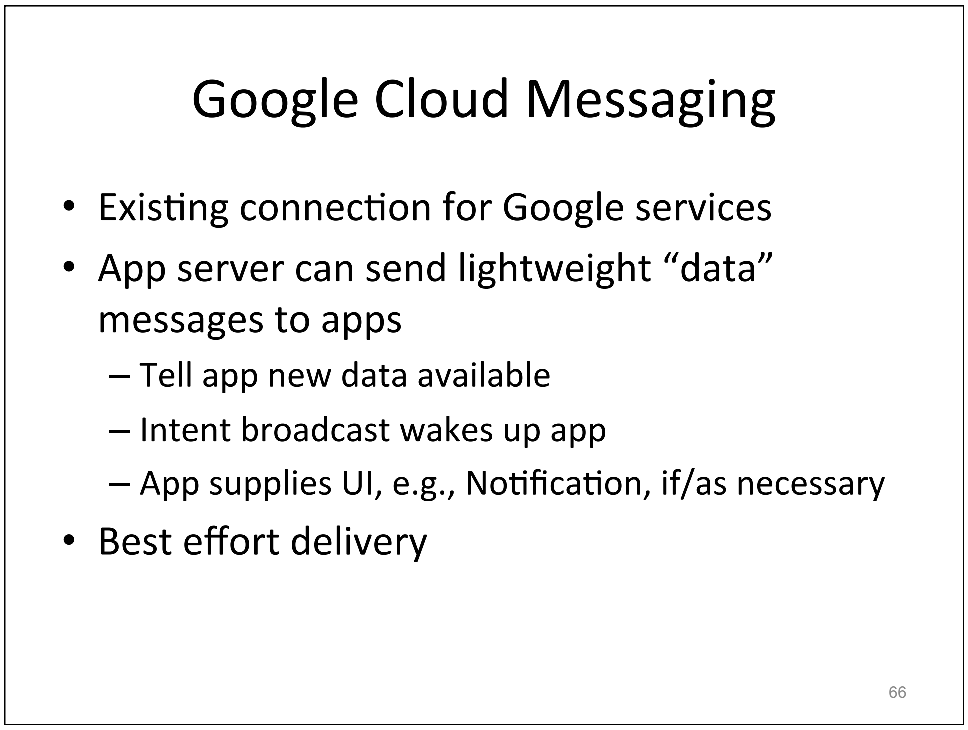
64



Pushing

- Reduce radio power drain
  - – Only use network when necessary
  - – Constant overhead of *persistent connection*
- Google Contacts, Calendar, Gmail, etc., use push sync
- Google Cloud Messaging (GCM)

65



Google Cloud Messaging

- Existing connection for Google services
- App server can 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

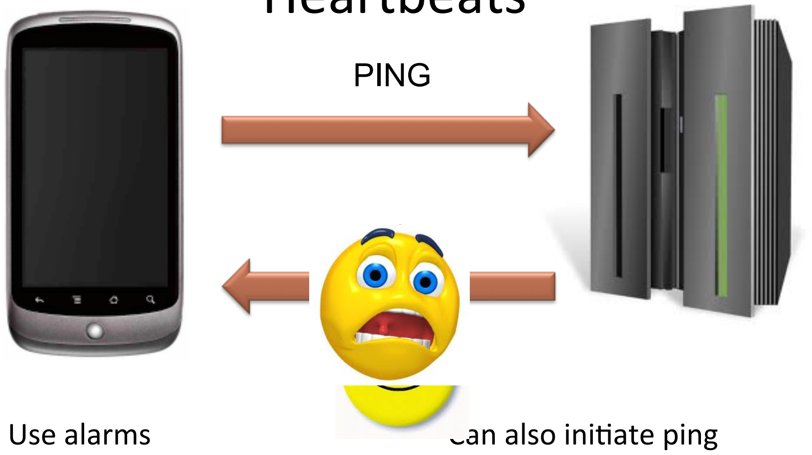
66

Peeking Under the Hood

- Background service
  - Honor background data setting
  - Start when network available
- Maintain connection with server
  - Heartbeats
  - Detect dead (“half-open”?) connections
- Efficient
  - Minimize per connect overhead
  - Minimize heartbeat frequency
  - Minimize concurrent connections

67

Heartbeats



- Use alarms
- Reconnect when dead

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

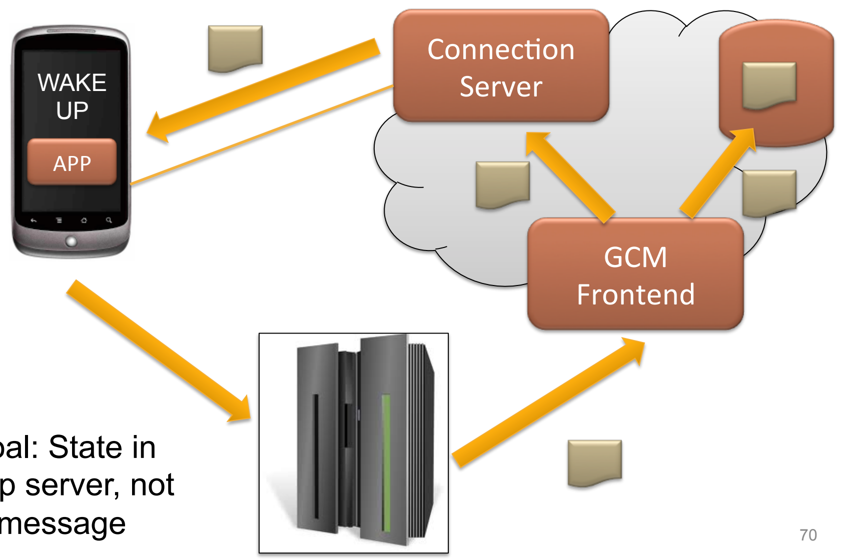
68

Overview of Lifecycle

- Enabling GCM
  - App (on device) gets registration ID
  - App sends registration ID to its App Server
- Per message
  - App Server sends (authenticated) message to GCM
  - GCM sends message to device
- Disabling GCM
  - GCM notifies App Server

69

Life of a Message



Goal: State in app server, not in message

70

Use Cases

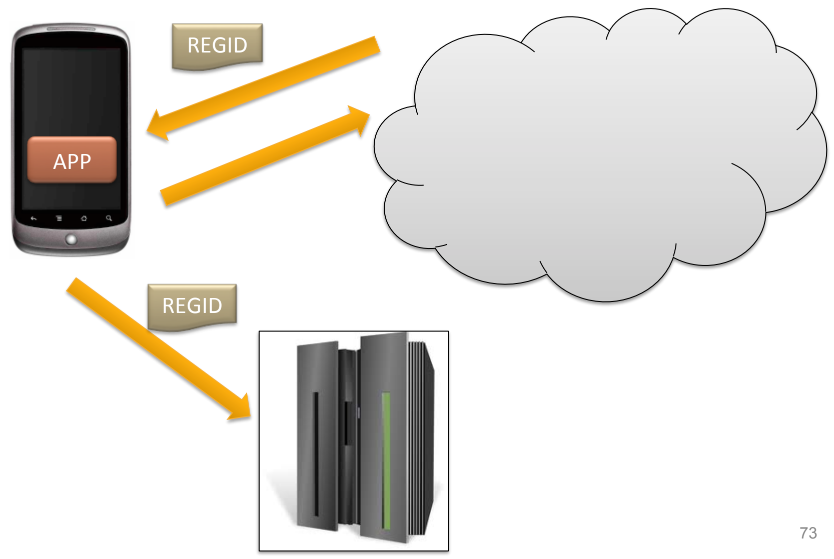
- Send-to-sync
- News
  - Multicast
- Events & promos
  - Time to live
- Instant message
  - Payload

71

GCM REGISTRATION

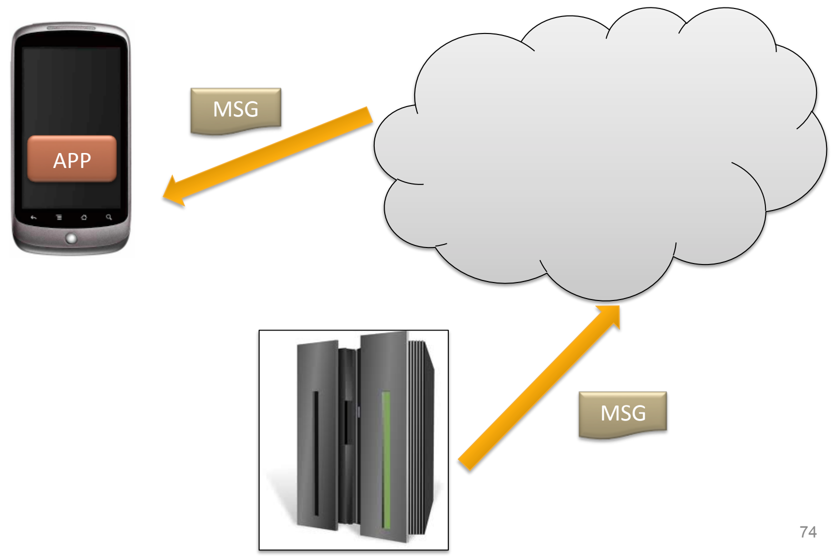
72

Registration



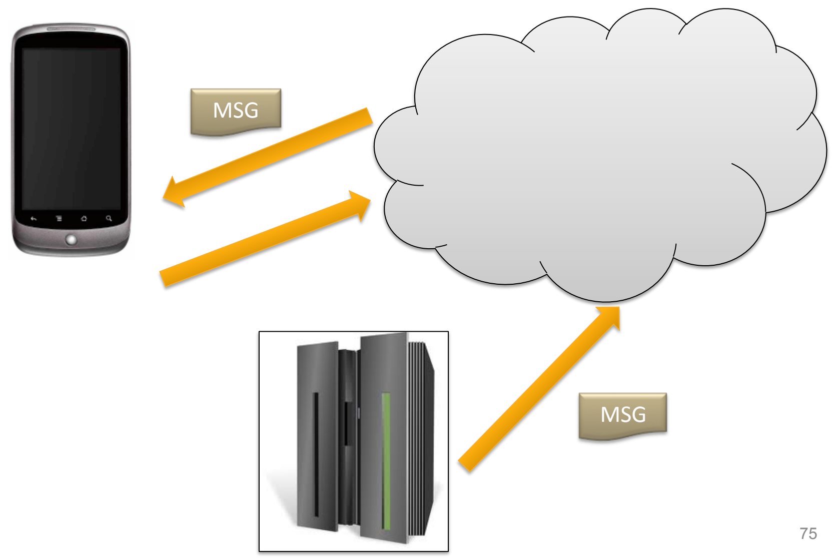
73

Sending



74

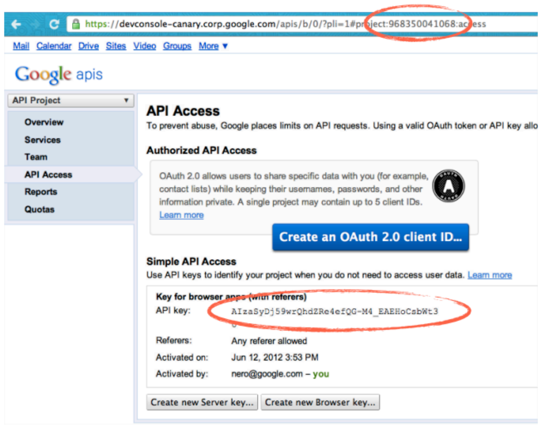
Unregistration



75

Registration

Auth


https://devconsole-canary.corp.google.com/apis/b/0/?pli=1#project:968350041068:access

Mail Calendar Drive Sites Video Groups More ▾

Google apis

API Project ▾
- Overview
- Services
- Team
- API Access
- Reports
- Quotas

API Access

To prevent abuse, Google places limits on API requests. Using a valid OAuth token or API key allo

Authorized API Access

OAuth 2.0 allows users to share specific data with you (for example, contact lists) while keeping their usernames, passwords, and other information private. A single project may contain up to 5 client IDs. Learn more

Create an OAuth 2.0 client ID...

Simple API Access

Use API keys to identify your project when you do not need to access user data. Learn more

Key for browser apps (with referers)	
API key:	AIzaSyDj59wrQhdZRe4efQG-M4_EAEHoCsbWt3
Referers:	Any referer allowed
Activated on:	Jun 12, 2012 3:53 PM
Activated by:	nero@google.com – you

Create new Server key... Create new Browser key...

76

Registration

- Credentials
  - Project key (userid)
  - API key (password)

```
import com.google.android.gcm.GCMRegistrar;

@Override
protected void onCreate(Bundle savedInstanceState) {
   GCMRegistrar.register(this, “968350041068”);
```

77

Registration

- Credentials
  - Project key (userid)
  - API key (password)

```
import com.google.android.gcm.GCMRegistrar;

@Override
protected void onCreate(Bundle savedInstanceState) {
   if (GCMRegistrar.getRegistrationId(this).equals(“”)) {
      GCMRegistrar.register(this, “968350041068”);
   }
```

78

Registration

- Credentials
  - Project key (userid)
  - API key (password)

```
import com.google.android.gcm.GCMBaseIntentService;

public class GCMIntentService extends GCMBaseIntentService {

@Override
protected void onRegistered(Context ctx, String regId) {
   sendToServer(regId);
```

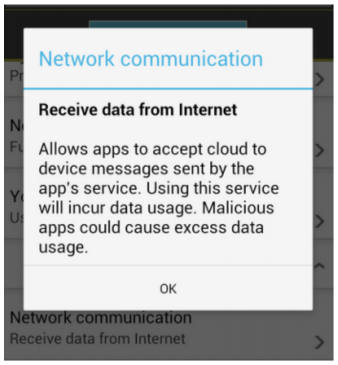
Receive Message

- Credentials
  - Project key (userid)
  - API key (password)

```
import com.google.android.gcm.GCMBaseIntentService;

public class GCMIntentService extends GCMBaseIntentService {

@Override
protected void onMessage(Context ctx, Intent intent) {
   final Bundle payload = intent.getExtras();
   ...payload...;
```

Permissions

```
<uses-permission
   android:name=
   "com.google.android.c2dm.permission.RECEIVE”
/>
```



Network communication

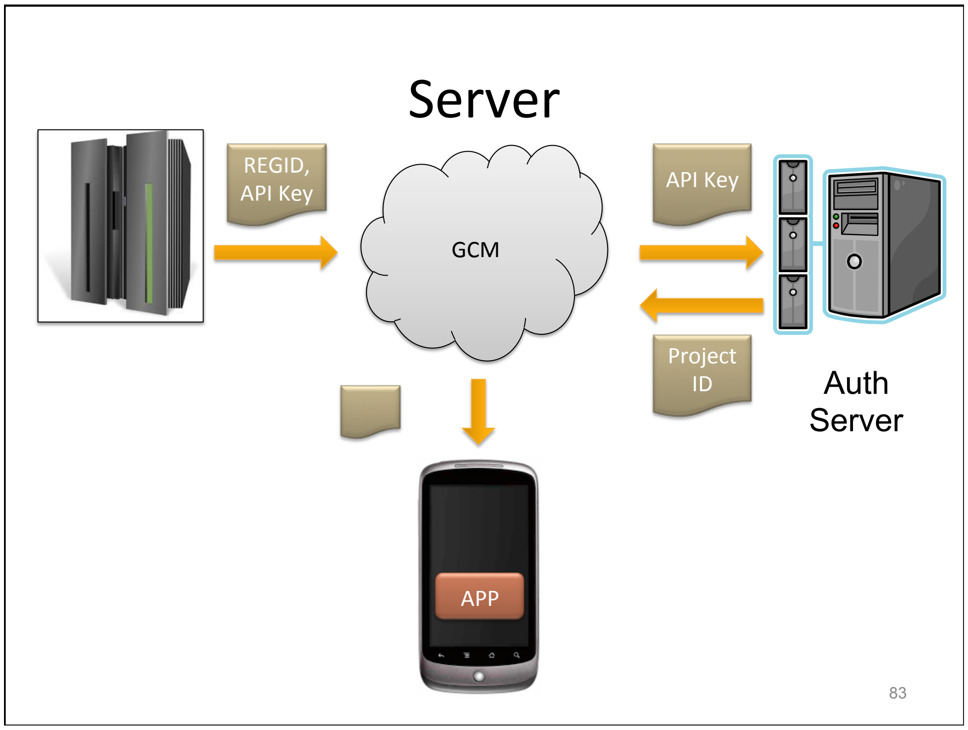
Receive data from Internet

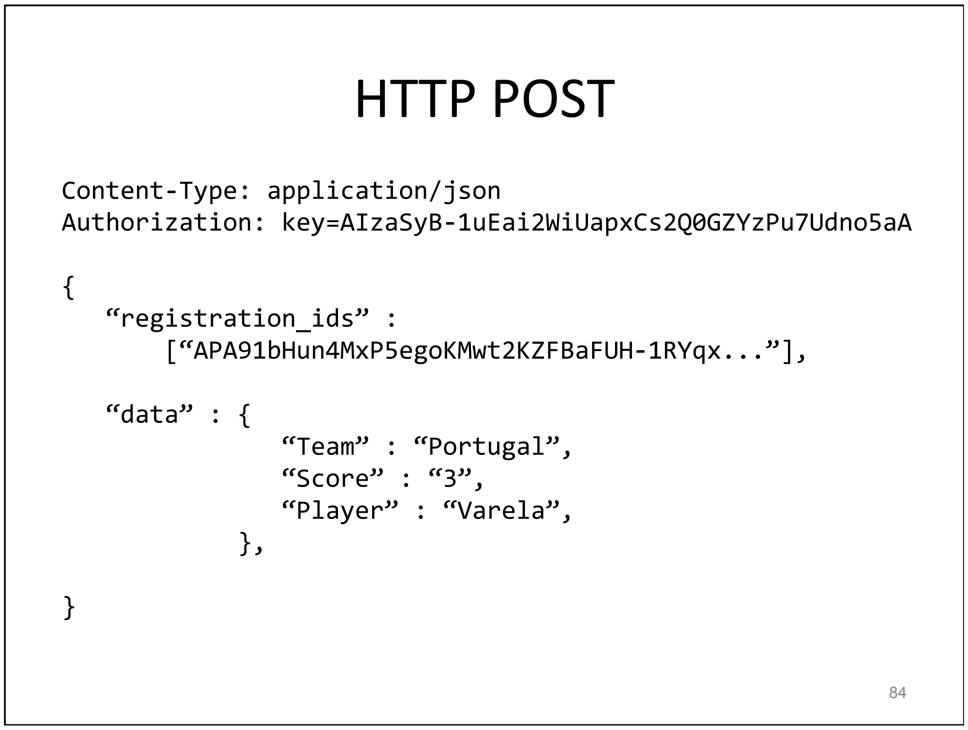
Allows apps to accept cloud to device messages sent by the app's service. Using this service will incur data usage. Malicious apps could cause excess data usage.

81

SERVER

82





HTTP POST

```
Content-Type: application/json
Authorization: key=AIzaSyB-1uEai2WiUapxCs2Q0GZYzPu7Udno5aA

{
   “registration_ids” :
       [“APA91bHun4MxP5egoKMwt2KZFBaFUH-1RYqx...”],

   “data” : {
                  “Team” : “Portugal”,
                  “Score” : “3”,
                  “Player” : “Varela”,
               },

}
```

84

Multicast

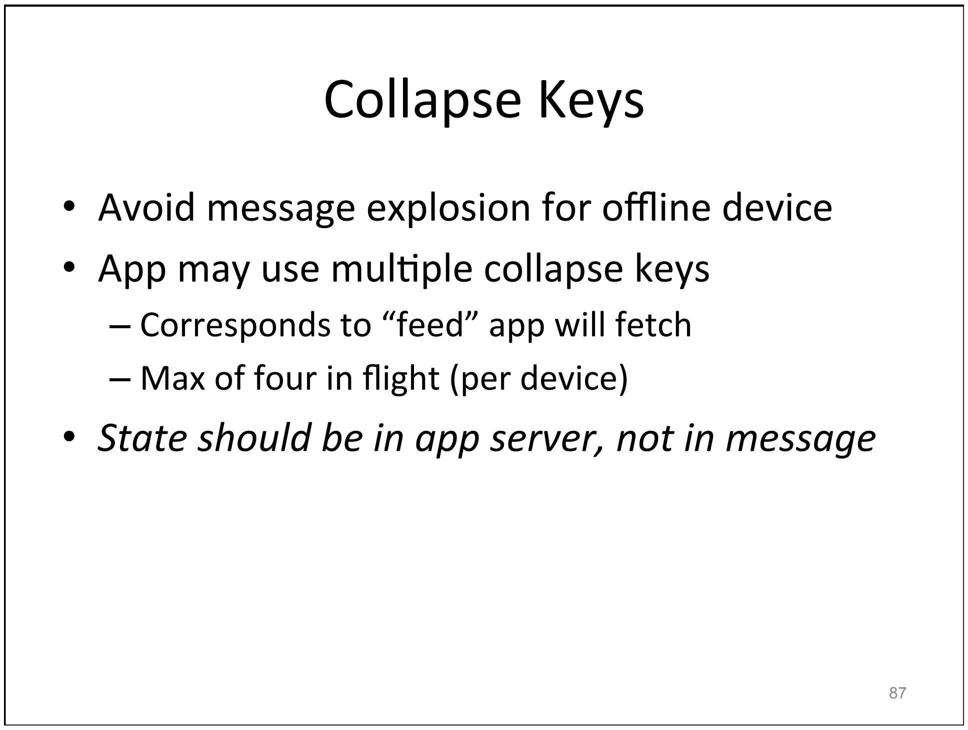
```
{
   “collapse_key” : “Beckham-News”,

   “data” : {
      “Team” : “LA Galaxy”,
      “Player” : “David Beckham”,
   },

   “registration_ids”:[
        “APA91bHun4MxP5egoKMwt2KZFBaFUH-1RYqx...”,
        “APQ23XFer5MtP0retKMfe1KSFWaFUH-1EWab...”,
   ]
},
```

85

Multicast Response

```
{
   “multicast_id” : “5814378600346514436”,
   “success” :1,
   “failure” :1,
   “results” :[
     { “message_id” :
       "0:1337639984251701%921c249af9fd7ecd", },
     { “error” : “DeviceNotRegistered”, }
   ],

}
```

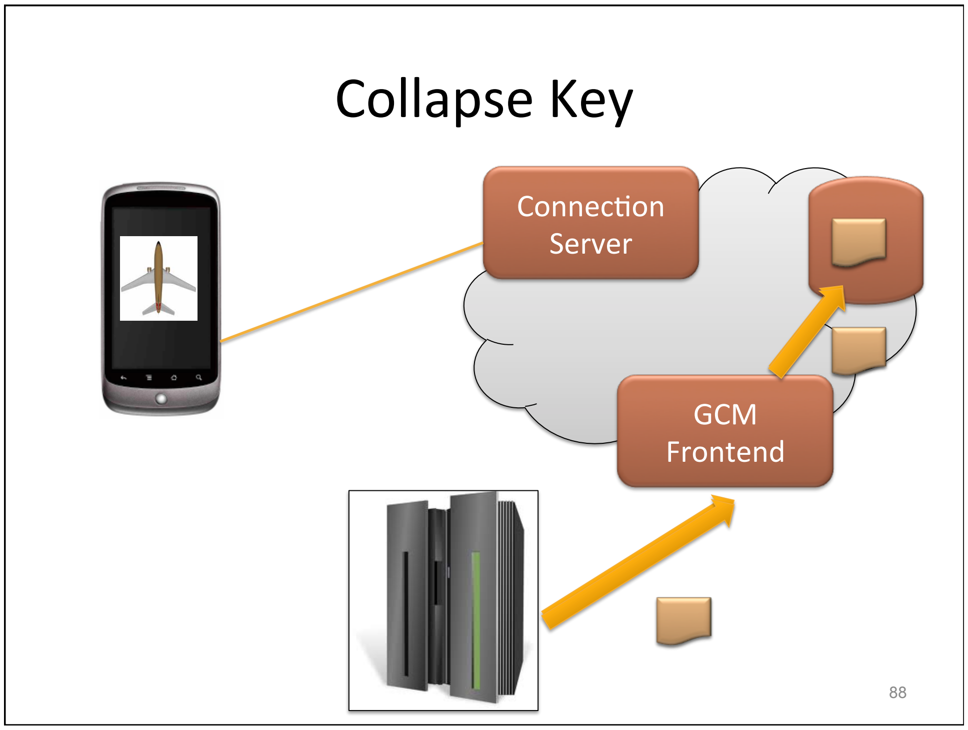
86



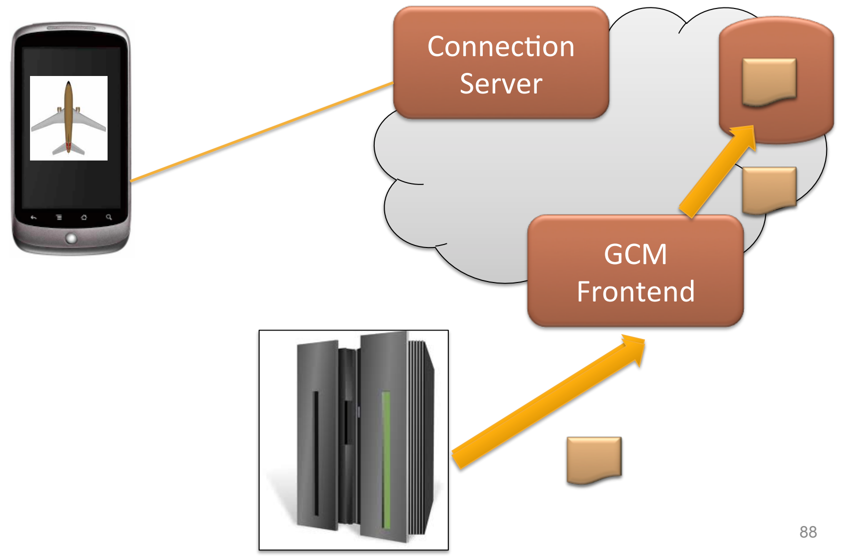
Collapse Keys

- Avoid message explosion for offline device
- App may use multiple collapse keys
  - Corresponds to “feed” app will fetch
  - Max of four in flight (per device)
- *State should be in app server, not in message*

87

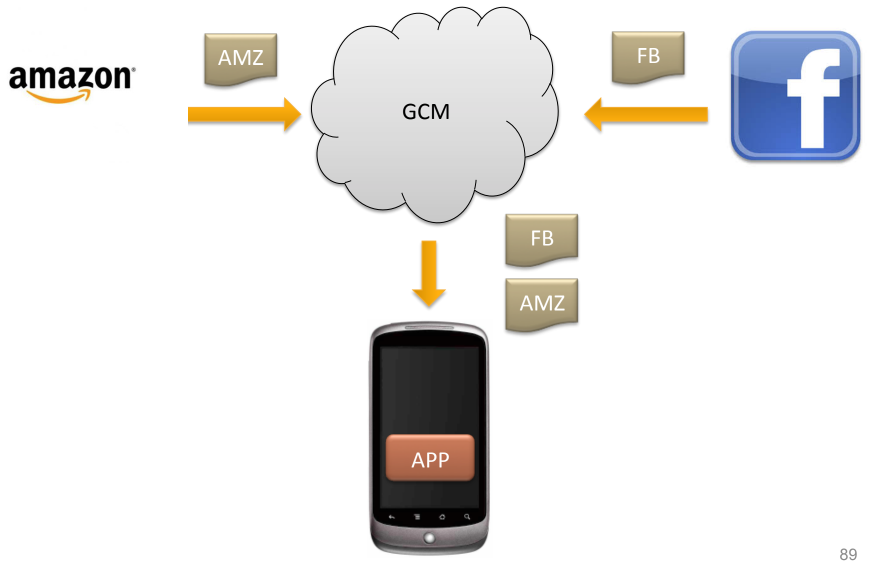


Collapse Key



88

Multiple (3^{rd} Party) Senders



89

Multiple (3^{rd} Party) Senders

```
import com.google.android.gcm.GCMRegistrar;

@Override
protected void onCreate(Bundle savedInstanceState) {

   if (GCMRegistrar
        .getRegistrationId(this).equals(“”)) {
     GCMRegistrar.register(this, “968350041068”,
                                 “652183961211”);
   }

 }
```

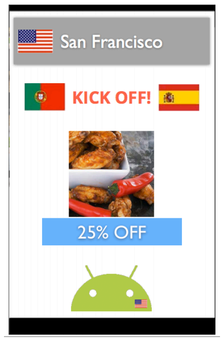
90

Multiple (3rd Party) Senders

```
import com.google.android.gcm.GCMBaseIntentService;

public class GCMIntentService extends GCMBaseIntentService {

@Override
protected void onRegistered(Context ctx, String regId) {
   sendToNewsServer(regId);
   sendToSocialNetwork(regId);
```

91

Time to Live



```
{
   “collapse_key” : “Food-Promo”,
   “time_to_live” : 3600,
   “delay_while_idle” : “true”,
   “data” : {
      “Category” : “FOOD”,
   }
   “registration_ids”:
   [“APA91bHun4MxP5egoKMwt2KZFBaFUH-1RYqx...”],

},
```

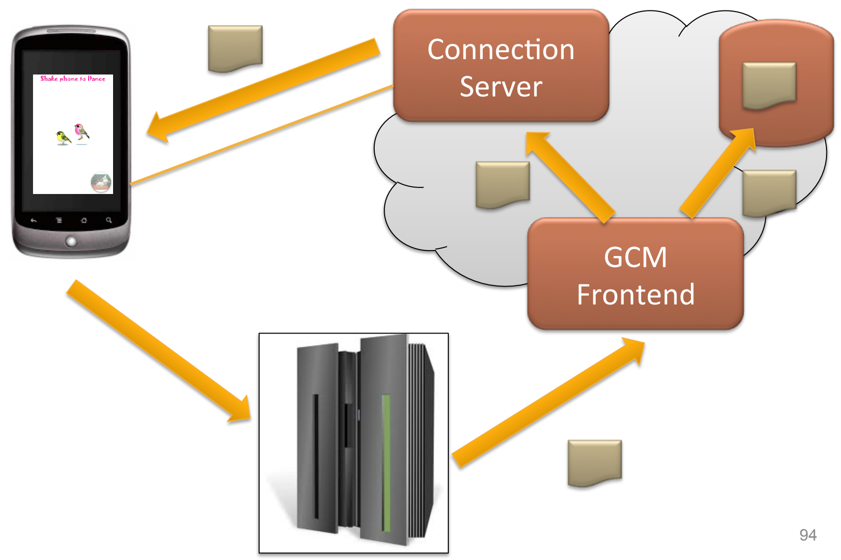
92

Delay While Idle

- Device tells Connection Server when screen is on, off
  - Device is idle?
- Apps can request message only be delivered when active
  - e.g., chat presence, friend location updates

93

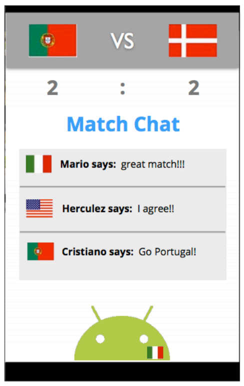
Delay While Idle



94

Messages with Payload

```
{
  “registration_ids” :
      [ “APA91bHun4MxP5egoKMwt2KZFBaFUH-1RYqx...”],
  “data” : {
      “Nick” : “Mario”,
      “Text” : “great match!”,
      “Room” : “PortugalVSDenmark”,
  },
}
```



Mario says: great match!!!

Herculez says: I agree!!

Cristiano says: Go Portugal!

95

Deleted Messages

- Device out of sync: delete messages
  - 512K
  - Force device to sync with server

```
{
 “message_type” : “deleted_messages”,
 “total_deleted” : “115”,
},
```

96

Deleted Messages

- Device out of sync: delete messages
  - 512K
  - Force device to sync with server

```
public class GCMIntentService extends

GCMBaseIntentService {

@Override
protected void onDeletedMessages(Context ctx,
                                 int total) {
   fullSyncWithServer(total);
}
```

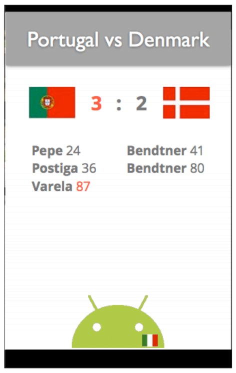
97

Summary

- Message multicasting for News
  - Max 1000 recipients
- Multiple senders for 3rd party
  - Max 100 senders
- Time to live for events and promos
  - 0 to 4 weeks
  - 4 weeks default
- Messages with payload for IM
  - Max 4K payload

98

Combining



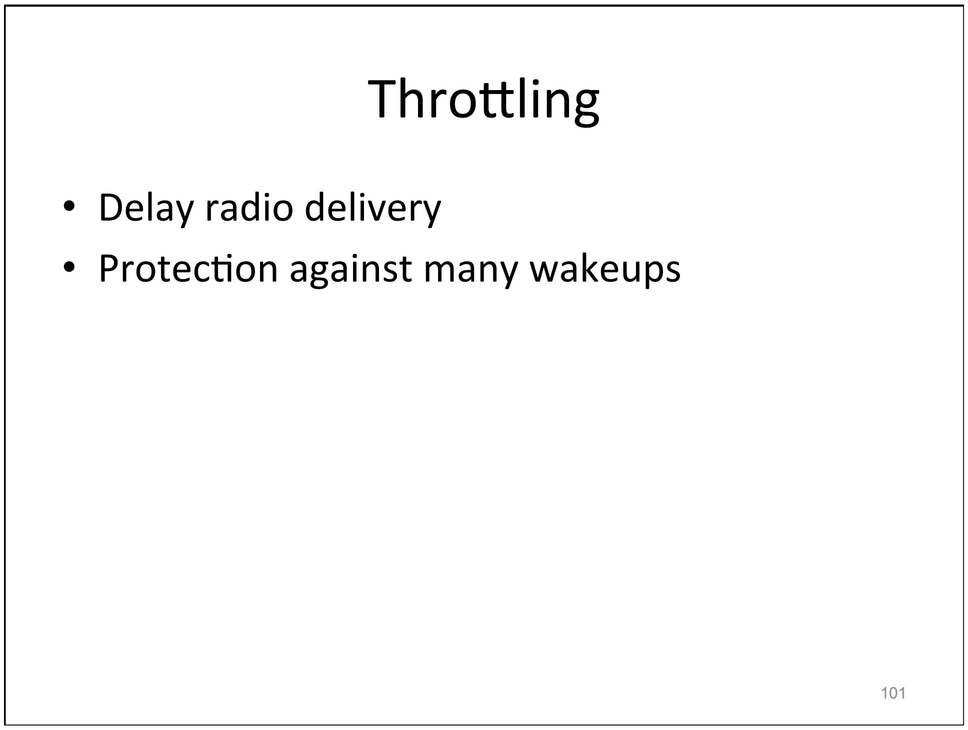
```
{
   “collapse_key” : “PortugalDenmark”,
   “time_to_live” : 4400,
   “data” : {
      “Team” : “Portugal”,
      “Score” : “3”,
   }

   “registration_ids”:[
      “APA91bHun4MxP5egoKMwt2KZFBaFUH-1RYqx...”,
      “APQ23XFer5MtP0retKMfe1KSFWaFUH-1EWab...”,
   ]
},
```

99

Reliability

- Reliable Message Queue
  - Messages queued at RMQ
  - Ack for every received message
  - Selective acks

100

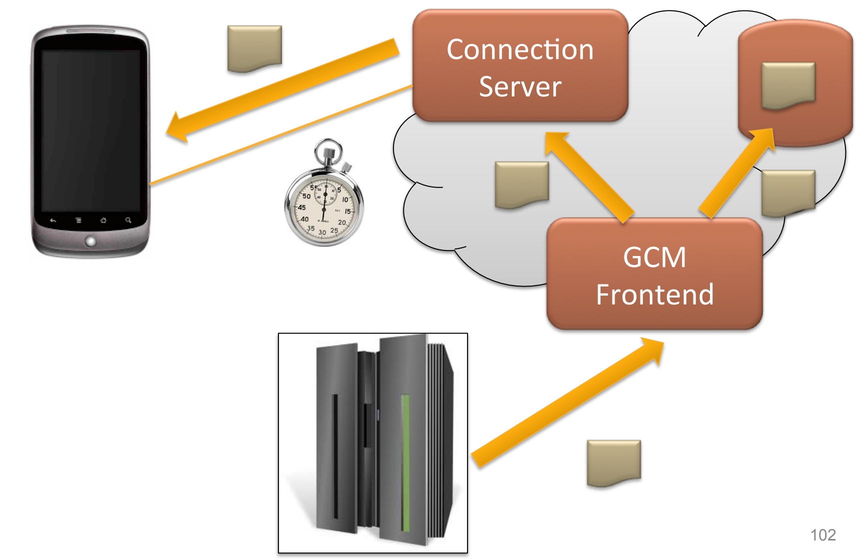


Throttling

- Delay radio delivery
- Protection against many wakeups

101

Throttling



102

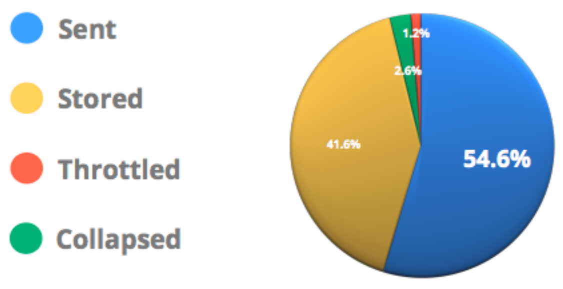
Throttling

- Combined throttle queue for all senders
- Piggyback messages on urgent message
- Throttle based on user wake-up
- Cooperation between device and GCM
  - When is device idle?
  - When to notify GCM of idle?

103

GCM Messages

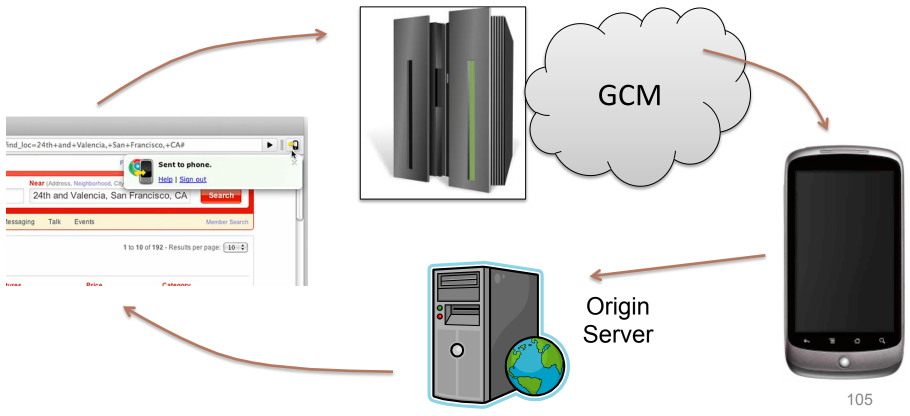
- Stored on device
- Stored on server
- Collapsed
- Throttled



104

Applications

- Chrome to Phone
  - Send web page, video etc to device



105