LMU LUDWIG-MAXIMILIANS-UNIVERSITÄT MÜNCHEN

Praktikum Mobile und Verteilte Systeme

Mobile Push Architectures

Prof. Dr. Claudia Linnhoff-Popien
Michael Beck, André Ebert
http://www.mobile.ifi.lmu.de

Sommersemester 2016

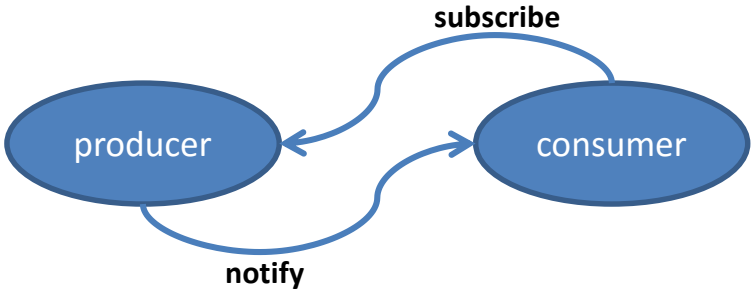
Asynchronous communications

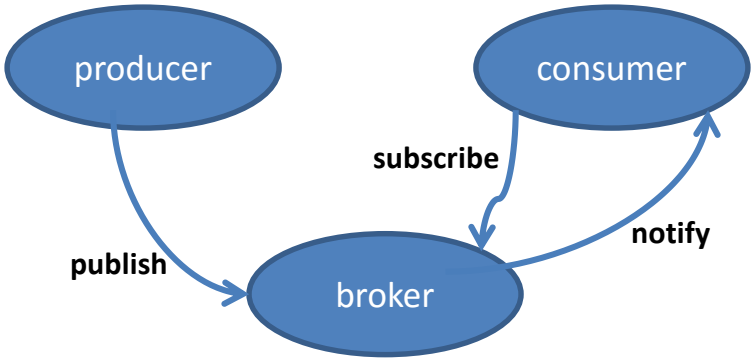
How to notify clients about changed resources or updates?

More general: How to **handle server-side events asynchronously**?

- **polling** is ineffective (e.g., continuously requesting a web service)

- SOAP offers **WS-Notification**
  - either peer-to-peer or brokered





- **Comet programming**: strategies for realizing push-like communication in pull-based environments (using HTTP)

Comet programming

- A web application model using persistent HTTP requests to push data to a browser
- Term coined by software engineer Alex Russell in a blog post in 2006
- First implementations date back to 2000
  - Pushlets, Lightstreamer, KnowNow
- In 2006, some widely known applications adapted these techniques
  - web-based chat application for AOL, Yahoo, Microsoft chat (Meebo)
  - Google: integration of a web-based chat in GMail
  - Comet-based, real-time collaborative document editing (JotSpot)

- Comet is an umbrella term, encompassing multiple techniques
  - relying on features included by default in browsers (e.g., JavaScript)
  - also known as Ajax Push, Reverse Ajax, Two-way-web, HTTP Streaming

Comet implementations

- **Streaming-based** implementations
  - Hidden iframe
    - uses chunked transfer encoding (no content-length) containing JavaScript tags
    - working in every common browser
  - XMLHttpRequest
    - server sends “multipart HTTP response” with each part invoking `onreadystatechange` callback
    - only working with few browsers

- **Long-polling** based implementations
  - XMLHttpRequest long polling
    - works like the standard use of XHR
    - an asynchronous request is sent to the server, response only after an update
    - after processing the response (or after a timeout), a new request will be sent
  - Script tag long polling
    - dynamically create script elements as `<src="cometserver/…js">`
    - payload contains new JavaScript events
    - cross-browser and cross-domain functionality

Mobile push architectures

- **Push notifications…**
  - are messages pushed to a central location and delivered to mobile devices
  - are comparable to the publish/subscribe pattern
  - often contain other technologies such as alerts, tiles, or raw data
  - offer an alternative to constantly polling data from servers

- These “central locations” are nowadays provided by Google, Apple, Microsoft, Blackberry, …

- Goal: **Push, don’t pull**
  - only fetch data when useful

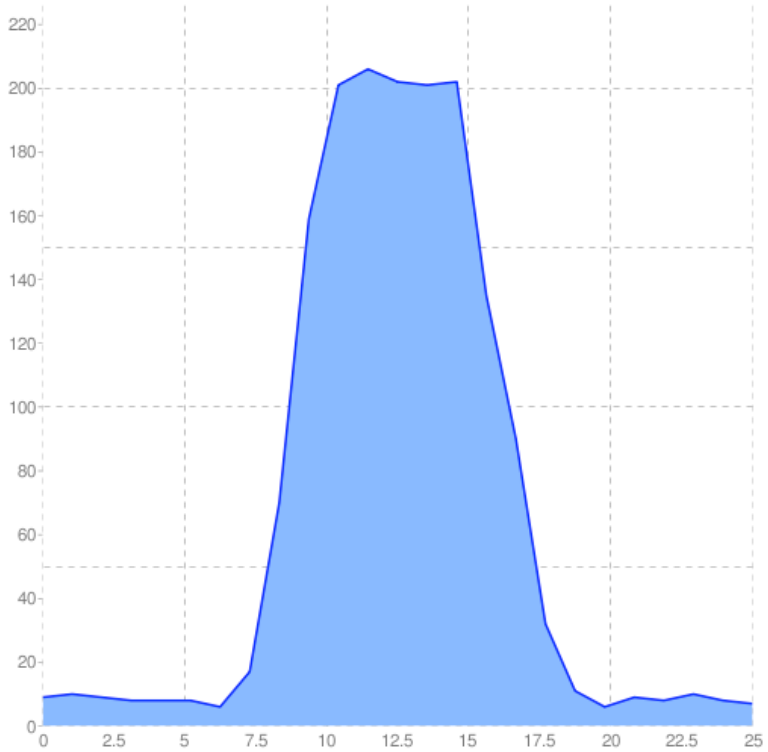
Advantages of push notifications (1)

Battery Life

- Baseline: 5-8 mA
- Network: 180-200 mA
- Radio stays on for few seconds
- 0.50 mAh for a short poll
  - 5m frequency: ~144 mAh / day
  - 15m frequency: ~48 mAh / day

- Push notification services are running in the background

- Pushing data is hence **more effective** than polling, if #updates < #polls



Source: Android development team at Google

Advantages of push notifications (2)

- **Message delivery and „time of flight“**
  - to save on battery, polls are usually spaced 15+ minutes apart
  - updated data might hence also be 15+ minutes late!
  - when using push notifications, message delivery can usually be expected to be a matter of seconds (<5s)
  - push notifications can also be sent to a currently offline device

- However, generally there is **no guarantee for delivery**
  - one might exceed quotas
  - some notification servers only allow a single message to be in queue at a time
  - …

Google C2DM

- The **Cloud to Device Messaging framework** allowed third-party servers to send lightweight messages to corresponding Android apps
- Designed for notifying apps about new content
- Makes **no guarantees** about delivery or the order of messages.
- Apps **do not have to be running** to receive notifications
  - the system will wake up the application via an Intent broadcast
- only passes raw data received to the application
- Requirements:
  - devices running Android 2.2 or above
  - have the Market application installed (Play Services)
  - a logged in Google account

- launched in 2010, officially deprecated as of June 26, 2012!
  - existing apps are still working, though

Google Cloud Messaging (GCM)

- successor of G2DM
- main differences:
  - to use the GCM service, you need to **obtain a Simple API Key** from the Google APIs console page
  - in C2DM, the Sender ID is an email address. In GCM, the **Sender ID** is a project number (acquired from the API console)
  - GCM HTTP requests **support JSON format** in addition to plain text
  - In GCM you can send the same message to multiple devices simultaneously (**multicast messaging**)
  - **Multiple parties** can send messages to the same app with one common registration ID
  - apps can send expiring invitation events with a **time-to-live** value between 0 and 4 weeks
    - GCM will store the messages until they expire
  - "**messages with payload**" to deliver messages of up to 4 Kb
  - GCM will store up to 100 messages
  - GCM provides **client and server helper libraries**

Google Cloud Messaging architecture (1)

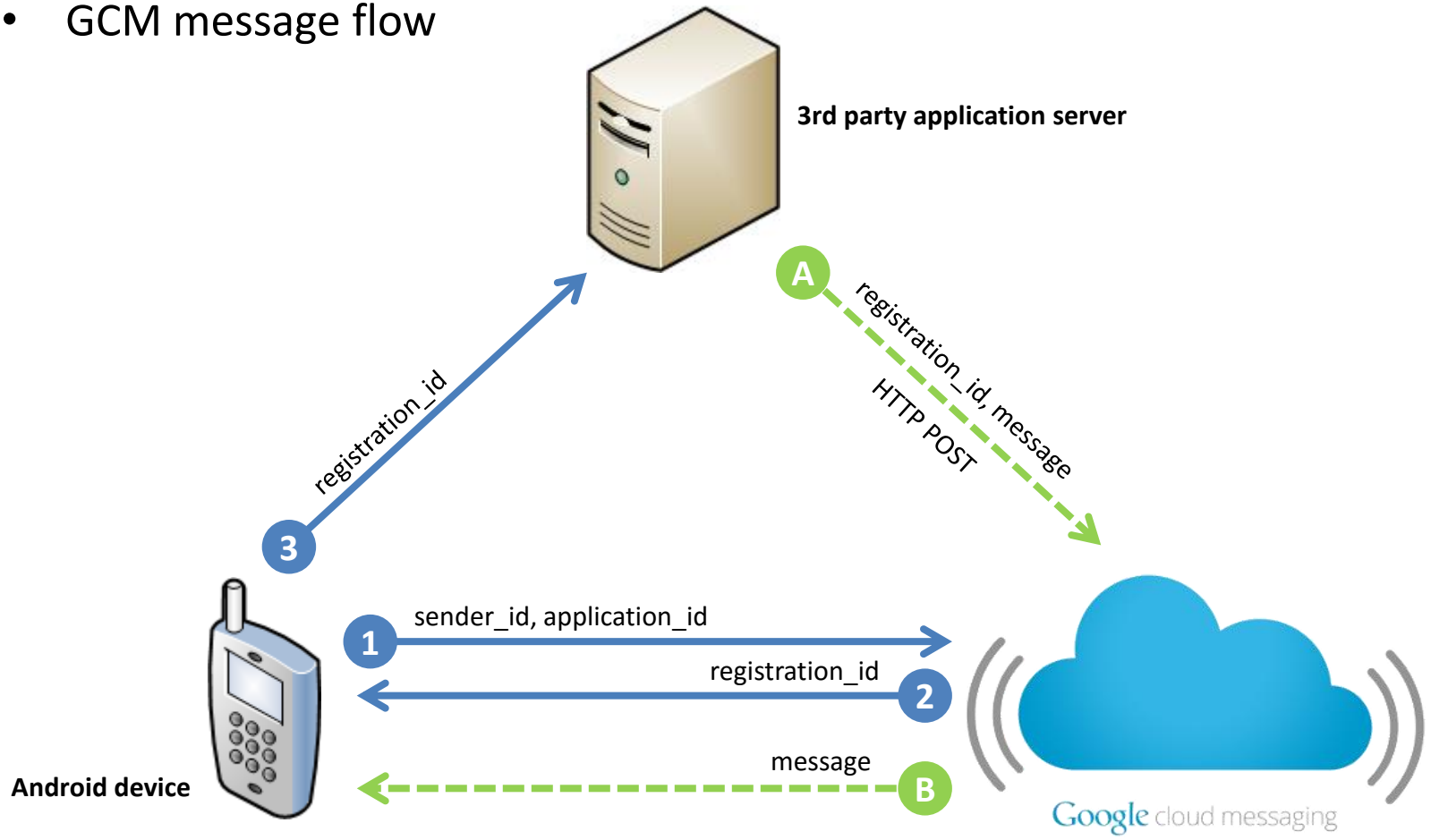
- GCM components
  - **Mobile Device** 
    - running an Android application that uses GCM
    - must be a 2.2 Android device that has Google Play Store installed
    - must have at least one logged in Google account
  - **3rd-party Application Server** 
    - a server set up by an app developer as part of implementing GCM
    - sends data to an Android application on the device via GCM
  - **GCM Servers** 
    - the Google servers involved in taking messages from the 3rd-party application server and sending them to the device

Google Cloud Messaging architecture (2)

- Credentials used in GCM
  - **Sender ID**
    - the project number (acquired from the API console)
    - used in order to identify the account that is permitted to send messages to the Android application
  - **Application ID**
    - used for identifying the application that is registering to receive messages (its package name as in the manifest file)
  - **Registration ID**
    - issued by the GCM servers to the Android application
    - used for identifying devices on the 3rd party server
  - **Google User Account**
  - **Sender Auth Token** (API key)
    - an API key stored on the 3rd-party application server
    - grants the application server authorized access to Google services

Google Cloud Messaging architecture (3)

- GCM message flow



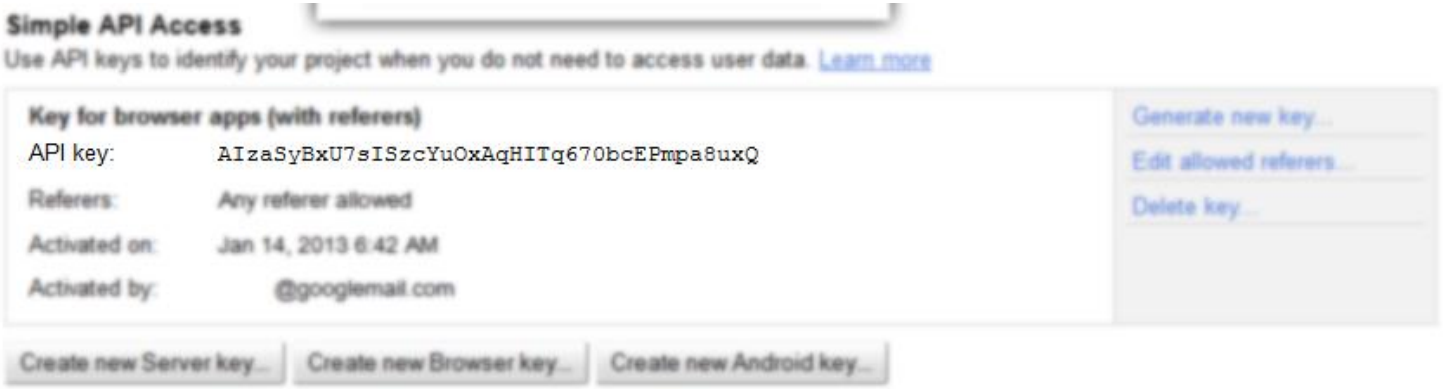
Using GCM with Java and Android (1)

- Create a **new Google API project** in order to get your SENDER_ID
  - Google APIs Console https://code.google.com/apis/console
  - https://code.google.com/apis/console/#project:**XXXXXXXXXX**

SENDER_ID

- **Enable GCM** services
  - Services → Google Cloud Messaging → ON

- Generate and find your **API key** (IP table might be empty)



Using GCM with Java and Android (2)

- Writing the **client application**
  - Download the **helper libraries**
    (SDK Manager, Extras > Google Cloud Messaging for Android Library)
  - Copy **gcm.jar** to your application's classpath
  - Adapt the Android **manifest file:**
    - **minSdkVersion** must be 8 or above
    - declare and use a **custom permission**, so that only your app will receive your push messages

```
<permission android:name="my_package.permission.C2D_MESSAGE"
        android:protectionLevel="signature" />
<uses-permission
        android:name="my_package.permission.C2D_MESSAGE" />
```

    - add **further permissions**:
      - `com.google.android.c2dm.permission.RECEIVE`
      - `android.permission.GET_ACCOUNTS`
      - `android.permission.WAKE_LOCK`

Using GCM with Java and Android (3)

- Writing the **client application**
  - add a broadcast receiver entry for `com.google.android.gcm.GCMBroadcastReceiver` (provided by the GCM library)

```
<receiver android:name="com.google.android.gcm.GCMBroadcastReceiver"
android:permission="com.google.android.c2dm.permission.SEND" >
  <intent-filter>
    <action android:name="com.google.android.c2dm.intent.RECEIVE" />
    <action android:name="com.google.android.c2dm.intent.REGISTRATION" />
    <category android:name="my_package" />
  </intent-filter>
</receiver>
```

  - add a `<service/>` entry for `.GCMIntentService`
  - implement `GCMIntentService` as subclass of `GCMBaseIntentService`
    - override at least its `onRegistered(), onUnregistered(), onMessage()` methods in order to be able to react to notifications

Using GCM with Java and Android (4)

- Writing the **client application**
  - handle notifications in the `onMessage` method

```
@Override
protected void onMessage(Context context, Intent intent) {
  String message = intent.getStringExtra("message");
  … // create a local notification (e.g., in the status bar)
}
```

  - in your main Activity, add something similar to this:

```
GCMRegistrar.checkDevice(this);
GCMRegistrar.checkManifest(this);
final String regId = GCMRegistrar.getRegistrationId(this);
if (regId.equals("")) {
  GCMRegistrar.register(this, SENDER_ID);
} else {
  Log.v(TAG, "Already registered");
}
```

Using GCM with Java and Android (5)

- Writing the **server-side application**
  - copy the **gcm-server.jar** to your server classpath
  - provide interfaces for registering and unregistering of devices
    - upon registration, a devices **registrationId** has to be stored
  - implement functionality for sending notifications to the registered devices when needed

```
Sender sender = new Sender("AIzaXXXXXXXXXXXXXXXXXXXXXXXXX");

Message message = new Message.Builder()
.collapseKey("1")
.timeToLive(3)
.delayWhileIdle(true)
.addData("message","sample text!")
.build();

Result result = sender.send(message,"device_token", 1);
```

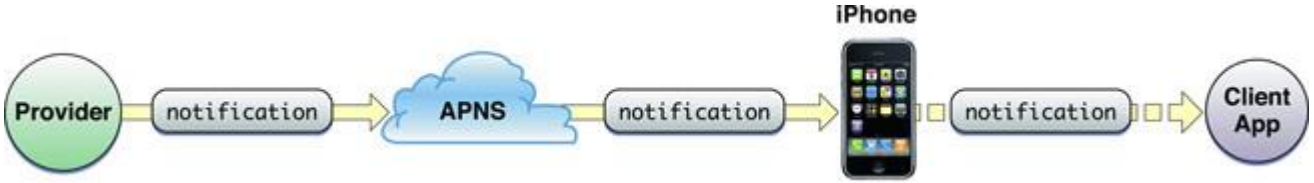
API_KEY

REGISTRATION_ID

#retries

Alternatives to GCM (1)

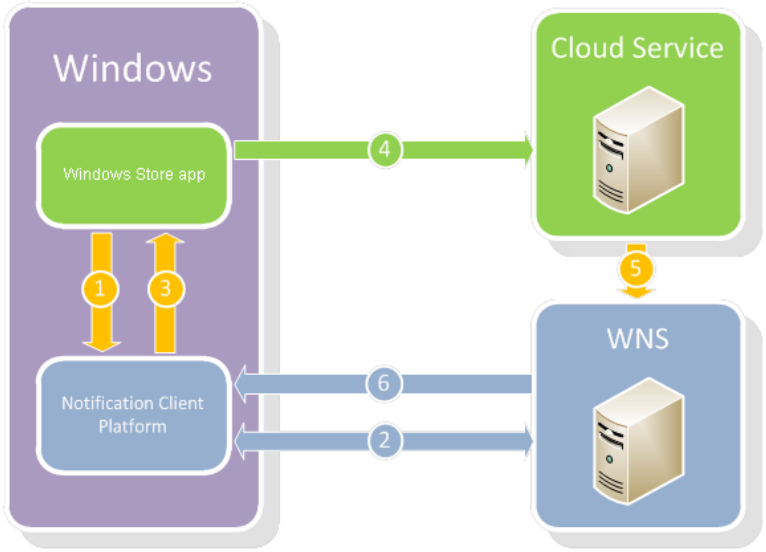
- **Apple Push Notification Service** (APNS)



  - launched with iOS 3.0 in 2009
  - maximum message size of 256 bytes, sent in JSON format
  - 3rd party servers can send lightweight notifications to apps
  - makes no guarantees for message delivery
  - making usage of alert messages, sounds and badges
    - iOS app does not have to be running

Alternatives to GCM (2)

- **Windows Push Notification Service** (WNS)
  - no delivery guarantee
  - enables third-party developers to send toast, tile, badge, and raw updates
  - message size up to 5kB



- Process:
  - app requests a push notification channel
  - this channel is returned to the calling device in form of a URI
  - the notification channel URI is returned by Windows to your app
  - app informs its application server about the URI
  - when the cloud service has an update to send, it notifies WNS using HTTP POST on the channel URI (SSL, requires authentication)
  - WNS routes the notification to the corresponding device

Further information

Google Cloud-Messaging (GCM)

- https://developers.google.com/cloud-messaging/

Apple Push Notification Service (APNS)

- https://developer.apple.com/library/ios/documentation/NetworkingInternet/Conceptual/RemoteNotificationsPG/Chapters/ApplePushService.html

Windows Push Notification Service (WNS)

- https://msdn.microsoft.com/de-de/library/windows/apps/mt187203.aspx

Praxis

- Übung 3 Bonusaufgabe