CASE d.o.o.

RAZVOJ POSLOVNIH
I INFORMATIČKIH SUSTAVA

# CASE 26

Srebrni pokrovitelji

foi

INFODOM

Brončani pokrovitelji

citus grupa
Internet Development Company

.1 ODJEL ZA INFORMATIKU
SVEUČILIŠTE U RIJECI

VELEUČILIŠTE U RIJECI
COLLEGIUM FLUMINENSIS

02.06.-03.06.2014, Zagreb

## ORGANIZATOR

**CASE d.o.o.**

## ORGANIZACIJSKI I PROGRAMSKI ODBOR

**TOMISLAV BRONZIN mag. ing. el.**
**ANTE POLONIJO**
**MISLAV POLONIJO**
**IVAN POGARČIĆ mr.sc.**
**ZLATKO SIROTIĆ univ.spec.inf.**
**ZLATKO STAPIĆ mag.inf.**

**Izdavač:**
CASE d.o.o., Rijeka

**Urednik:**
Mislav Polonijo

**Priprema za tisak:**
CASE d.o.o., Rijeka

**Tisak:**
CASE d.o.o., Rijeka

ISSN 1334-448X
UDK 007.5 : 621.39 : 681.324



Case d.o.o., Šetalište XIII divizije 28, 51000 Rijeka
tel: 051/217-875, tel/fax: 051/218-043, e-mail: case@case.hr, internet: www.case.hr

# S A D R Ž A J

## CASEdev



## CASEmobile

# KORIŠTENJE GOOGLE CLOUD MESSAGING SERVISA U ANDROIDU

## USING GOOGLE CLOUD MESSAGING SERVICE IN ANDROID

**Zoran Kos, Zlatko Stapić**


## SAŽETAK

*Rad prikazuje mogućnosti korištenja Google Cloud Messaging (GCM) servisa za Android platformu, te koji omogućuje developerima da pomoću web servisa šalju notifikacije i podatke instaliranim aplikacijama, na pametnim i mobilnim uređajima ili tabletima. GCM pritom vodi brigu o stanju u kojem se mobilni uređaj nalazi, a asinkrona komunikacija je moguća i u suprotnom smjeru, to jest od mobilne aplikacije prema servisu. GCM raspolaže s mnogo korisnih značajki poput „send to sync" i „senddata" mogućnosti, podrške za multicast poruke, rada u slučaju mirovanja, uporabu perzistentne veze (upotreba XMPP-a), upstream slanja poruka i sinkronizacije obavijesti na većem broju mobilnih uređaja. Također ova usluga se može iskoristiti za lociranje ukradenih telefona, daljinsko podešavanje telefona, slanje poruka prilikom postizanja određene razine u igrama itd. Ukratko, usluga GCM je odličan način za programere da upravljanju s provjerenim aplikacijama, ali sa sobom nosi i određene nedostatke jer je u zadnje vrijeme postala predmet računalnog kriminala.*

## ABSTRACT

*This article presents the possibility of using Google Cloud Messaging (GCM) service for the Android platform. GCM provides programmers and developers the possibility to send notifications and data to smart phones' or tablet's installed applications, by means of using the Web services. It also takes care of the state in which the mobile device, and gives a possibility of asynchronous communication in the opposite direction – from mobile application to the service. GCM has many useful features such as: "Send to Sync" and "Send data" capabilities, support for multicast messages, delay while idle, the use of persistent connections (by using XMPP), the upstream messaging and synchronization notification across multiple android devices. Also, this service can be very useful for locating stolen phones, remote adjustment of the mobile device sending a messages when user reach a certain level in the game, and so on. Shortly, the GCM is a great way for Android developers to manage their applications, but it also has its disadvantages and it recently became the subject of cyber-attacks.*


## 1. INTRODUCTION

Google cloud messaging (GCM) for the Android platform is a service that provides applications' programmers and developers to communicate with applications installed on smart phones, tablets, Chrome apps and extensions, from servers. GCM was officially presented in 2012. at Google I/O conference in San Francisco, California. The next year, at the same conference, new improvements were introduced (Google Developers, 2013). Through this service, it is possible to send a wide range of information, from


*Figure 1 - GCM life cycle flow (HMKCode, 2014)*

ordinary notifications to the commands that are executed inside the application by providing fresh and up-to-date data for users. This service has been developed to assist developers in building apps whose communication or data transfer goes asynchronous in two directions, between service and applications on the same connection (push notification). This data could be a lightweight message telling the Android application that there is a new data to be fetched from the server (for instance, a "new email" notification informing the application that it is out of sync with the backend), or it could be a message containing up to 4KB of payload data (so apps like instant messaging can consume the message directly) (Android Developers, 2014a). Google uses GCM for the processing and sending instant notifications for many popular applications including Twitter, Facebook and Gmail. This applications use GCM to inform the users about: the friends that have checked in nearby; the received a message; any related notice with user profile.

The strength of this service can be seen in the fact that it can serve more than 200,000 push notification every second. This paper describes how the messaging works, and it consists of five sections. After this introduction the second section describes the main characteristics and components of which GCM is composed. The third chapter describes initial steps that have to be made on the client and the server side implementation of the GCM. It also describes what are user notifications and their role in the GCM. The fourth chapter shortly writes about the vulnerabilities and attacks that were recorded on GCM. At the end of this paper conclusion is given.

## 2. THE MAIN GCM CHARACTERISTICS

The GCM architecture is graphically presented in Figure 1 and it contains the three following main components:

- **Client Application** – receives the message from GCM connection server initially sent by the application server and sends the new message to the same server.
- **3rd-party Application server** – receives the client app registration ID; based on the registration ID records in the database sends the message to the GCM connection server.
- **GCM connection server** – receives the messages from the application server and sends these messages to the GCM enabled android devices.

Before using GCM there was the Android Cloud to Device Messaging (C2DM) service that helped sending data from servers to applications on the mobile devices. But with appearance of GCM, C2DM has been officially deprecated, meaning that the C2DM stopped accepting new users, denies new quota requests and has no new available features. The reason for deprecation was that GCM can be simply put to use. There is no signup form; no quotas or client login token, it takes 4.7 ms to deliver the messages, and all of this reduces battery usage and gives a rich set of new APIs. GCM also provides a client and server helper library which means that developers can easily write code. The GCM service is completely free and handles all aspects of queuing of messages and delivery to the target Android application running on the target device. Considering these changes it is not possible to establish interoperability between GCM and C2DM. All GCM major features that are new and make a difference in relation to C2DM are placed in Table 1:

| GCM features and APIs |
|---|
| Simple API key |
| Sender ID |
| Canonical registration ID |
| Json format |
| Multicast messages |
| Time-to-live messages |
| Messages with payload |
| Send-to-Sync messages |
| Upstream messaging |
| Seamless multi-device messaging |
| Multiple senders |

*Table 1 – GCM features and APIs*

Key Concepts of GCM are components, entities of GCM architecture and the credentials used for access authentication in different stages of GCM. Credentials can ensure that all components have been properly authenticated so that data transfer and messaging is going to the correct place and without delay. Credentials are the IDs and tokens used in different stages of GCM which are listed below (Android Developers, 2014a):

- Sender ID – a project number given from API console, used in the registration process to identify a 3rd-praty app server.
- Application ID – located in manifest and identified by the package name. This ID ensures that the messages are targeted to the correct app.
- Registration ID – issued by the GCM servers to the app that allows it to receive messages. This ID is tied to a particular app running on a particular device. 3rd-party app server can identify each device that has registered to receive messages for a given app. If client app no longer wants to receive messages it can unregister GCM.
- Google User Account – used only if use's Android API is lower than 4.0.4.
- Sender Auth token – is an API key that is saved on the 3rd-party app server. It gives app server authorized access to Google services. The token is placed in the header of POST requests that send messages.

If we put attention on Figure 1 we can see the lifecycle flow of GCM which consists of three processes that are described in more detail below:

- Enable GCM
- Send a message
- Receive a message

To enable GCM, client app that runs on mobile device calls register method which returns the register ID in order to receive messages. A 3rd-party app server sends messages to the device, and this process consists of several steps (Android Developers, 2014a):

- Google checks and stores the message in case the device is offline.
- When the device is online, Google sends the message to the device.
- The system broadcasts the message to the specified app via Intent broadcast with proper permissions, so that only targeted app gets the message.
- App processes the message.

Process *"receive a message"* consists of a sequence of steps starting from extracting the raw key/value pairs from the message payload. After that system transmits this key/value pairs to the targeted app as a set of

extras to broadcast receiver. App then extracts raw data from this intent and processes the data. The app server sends a message to GCM servers. The described processes make the life-cycle of GCM which is shown in Figure 2:

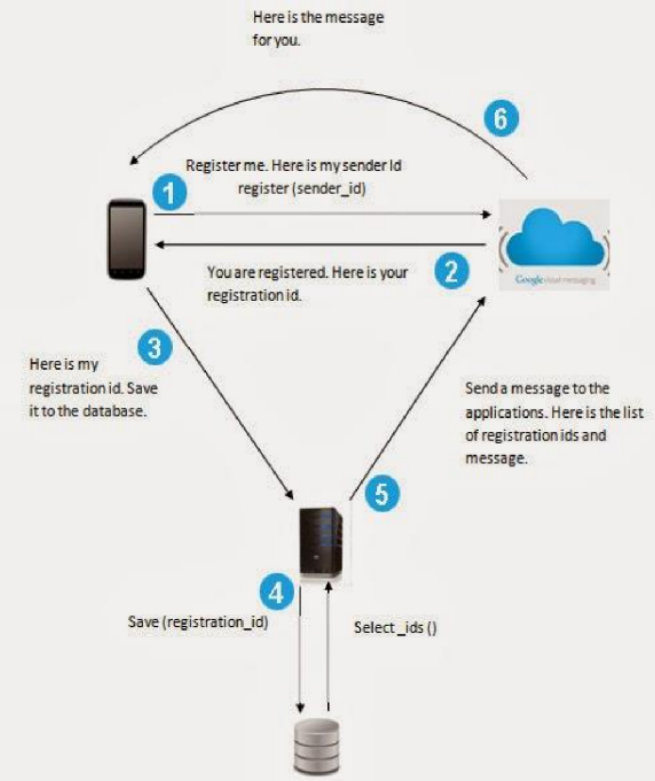


*Figure 2 – GCM Life Cycle Flow (freelancersnepal, 2014)*

## 3. IMPLEMENTATION OF THE GCM

This chapter provides short guidelines, the developers should hold, in order to successfully implement the GCM. Before GCM can be used in java project, some steps should be taken. They are listed and explained below:

1 Installation of Google Play services SDK – provides the GoogleCloudMessaging methods and it can be used in combination with Eclipse ADT environment.
2 Creating a Google API project inside the Google Developers Console – it creates project ID and project number.
3 Enabling the GCM service API inside created project.
4 Creating a Server key – with option to whitelist specific IP addresses. This key is used by developer's server app as a password when connecting to Google's Command and Control server.

After successfully completing above steps GCM project in the API Console is created and it's possible to start implementing GCM. Developers can thereby decide which of the available GCM connection servers will be used (HTTP or XMPP). Then, it is possible to start implementing an app server to interact with chosen GCM connection server or GCM-enabled app that runs on a device. Those who do not have own available 3rd-party app server can use upstream messaging (device-to-cloud) feature or user notification via **Command and Control server** (**CCS** is an XMPP endpoint that provides a persistent connection to Google server) by becoming trial partner with Google available at given URL: *https://services.google.com/fb/forms/gcml/.*

### 3.1 How to implement GCM Client

As mentioned earlier in the previous chapter, the GCM client is GCM-enabled app that runs on mobile device which uses `GoogleCloudMessaging` API. It is also possible to use the *client helper library* that was offered in previous version of GCM but now it's not recommended to be used because some methods in this library are deprecated. The following example shows how to implement GCM client in mobile device that runs on Android platform. For purposes of better presentation and programming, the next example parts of the codes are taken and modified from publicly available pages (tutorials) which can be found at (Android Developers, 2014b) and (Cambell, 2013). To implement a GCM client-side application, application must include code to register (Registration ID) and broadcast receiver to receive messages sent by GCM. In order to do that we can use the form shown in Fig. 3:

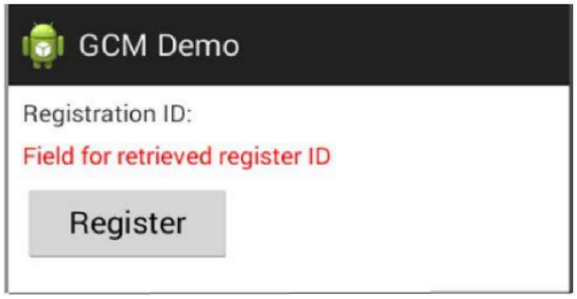


*Figure 3 – Registration application form*

Starting point is to include `GoogleCloud Messaging` API (available within Google play services SDK) into the project. Without this API it is not possible to implement GCM on client side. The second point is writing permissions in the application manifest:

- `com.google.android.c2dm.permission.RECEIVE` – app can register and receive messages.
- `com.google.android.c2dm.intent.`
- `RECEIVE` – if set, only the GCM Framework can send a message to it.
- `android.permission.INTERNET` – for sending registration ID to the 3rd party server.
- `android.permission.GET_ACCOUNTS` – if mobile device is running on lower API then Android 4.0.4, GCM must require a Google account.
- `android.permission.WAKE_LOCK` – client app can keep the processor from sleeping when a message is received.
- `applicationPackage.permission.`
- `C2D_MESSAGE` – prevents other apps from registering and receiving the GCM message.

For a better understanding it is necessary to pay attention to *receive permission*. Because, if client app uses an `IntentService` (class for Services that handle asynchronous requests on demand), broadcast receiver should be an instance of `WakefulBroadcastReceiver` (creates and manages partial wake lock on client app). A `Service` (typically an `IntentService`) to which the `WakefulBroadcastReceiver` passes the work of handling the GCM message, while ensuring that the device does not go back to sleep in the process codes like this (Android Developers, 2014b):

```
<receiver
android:name=".GcmReceiver"
android:permission="com.google.android.
                    c2dm.permission.SEND"
>
<intent-filter>
     <action android:name="com.google.
android.c2dm.intent.RECEIVE" />
     <category android:name="com.
case26.gcmdemo" />
   </intent-filter>
 </receiver>
 <service android:name=
".GcmIntentService" />
```

*Code 1 – Enabling send messages*

Broadcast receiver performs operations automatically when *service* starts with operations. The following code snippet starts `GcmService` with the method `startWakefulService()` inside the `GcmReceiver` class:

```
// Explicitly specify that GcmService
                  will handle the intent.
ComponentName comp=new
   ComponentName(context.getPackageName()
            ,GcmService.class.getName());

// Start the service, keeping the device
             awake while it is launching.
startWakefulService(context,
            (intent.setComponent(comp)));
setResultCode(Activity.RESULT_OK);
```

*Code 2 – Starting GCM Service*

Client app cannot receive messages before it is registered with the GCM servers. To achieve this, developers must use class that extends the `GoogleCloudMessaging` class with `AsyncTask object` so receiving process can be performed in the background. `AsyncTask class` enables easy use of the UI thread and performing background operations. Usually, registration should occur in the `onCreate()` method in main activity or using a dialog if we want to give the user a choice about receiving GCM messages. It is necessary to include values of variables `Sender_ID and Application_ID` given by Google Developers Console. Once the registration ID is retrieved it is stored locally on the device (`SharedPreferences`) and sent to 3rd party server. It is recommended to use HTTP connection so developers can have an immediate answer if the registration went well or not. The following code snippet shows short registration process, described previously:

```
if(gcm == null){
gcm=GoogleCloudMessaging.
getInstance(context);
}
regID=gcm.register(Globals.
  GCM_SENDER_ID);

// Send the registration ID to 3rd part
                  server server via HTTP
sendRegistrationIdToBackend();

// PersisttheregID - no need to
                          registeragain
storeRegistrationId(context, regid);
```

*Code 3 – Registers the app with GCM server asynchronously*

If the registration went well with no errors as the feedback of Figure 4 on the screen of demo application will be show the following:

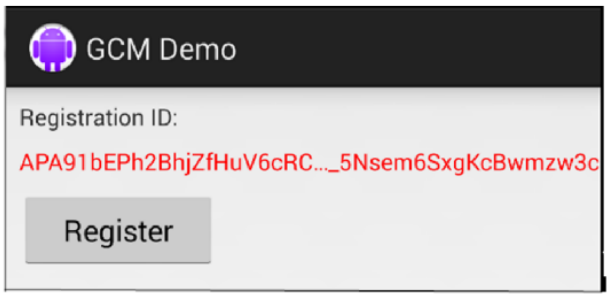


*Figure 4 – retrieved Register ID*

Based on the obtained Register ID, client app can also send messages to 3rd party application server over GCM server. The content of the message depends on what the developer wants to get out of the user's activities. It is important to point out that the GCM message is a JSON object containing some required and optional fields. There are several different formats depending on the direction in which communication goes:

- **Request format** - from a 3rd-party app server or client app to CCS.
- **Response format** – from CCS to 3rd-party app server (including ACK and NACK message formats).

For each device message that the 3rd-party application server receives from CCS it needs to send an ACK message. 3rd-party application server never sends a NACK message. CCS sends an ACK or NACK for each server-to-device message, if client app doesn't send an ACK for a message; CCS will just resend it again. The following code snippet shows how request format of JSON massage should look like:

```
"to":"REGISTRATION_ID",
"message_id":"m-453443534546",
"data":
{
"message":"case26",
"action":"com.case26.gcmdemo.ECHO",
}
"time_to_live":"500",
"delay_while_idle": false
```

*Code 4 – JSON massage, Request format*

We could get specified format with following code snippet which is written in java GcmActivity class:

```
 newAsyncTask(){
   @Override
   protectedStringdoInBackground
   (Void...params){
   String msg = "";
    try{
    Bundle data = newBundle();
       data.putString("my_message",
      "case26");
   data.putString("my_action",
```

```
    com.case26.gcmdemo.ECHO ");
    String id =Integer.toString
    (msgId.incrementAndGet());
    gcm.send(SENDER_ID +"@gcm.
    googleapis.com", id, data);
       msg="Sent message"; }catch
                          (IOException ex){
           msg="Error :"+ex.getMessage();
           }
           return msg;
    }
}
```

*Code 5 – Upstream a GCM message up to the 3rd party server*

### 3.2 How to implement GCM Server

As mentioned previously, server side of GCM consists of Google-provided GCM Connection Servers (HTTP or CCS over XMPP) and 3rd-party application server which developer must implement. 3rd-party server must be able to communicate with client and GCM Connection Servers with properly formatted requests and be able to respond incoming requests to it. It's also important that each exchanged massage have a unique message ID that is located in the header of POST request. The first step that developer has to do is to make a decision which type of GCM connection server(s) to use. For a better understanding differences between two types of connection server(s) through the basic features are listed (Android Developers, 2014c):

- **Upstream/Downstream messages:**
  **HTTP:** Downstream only: cloud-to-device
  **CCS:** Upstream and downstream (device-to-cloud, cloud-to-device)
- **Asynchronous messaging**
  **HTTP:** 3rd-party app servers send messages as HTTP POST requests and wait for a response
  **CCS:** 3rd-party app servers connect to Google infrastructure using a persistent XMPP connection and send/receive messages to/from all their devices at full line speed.
- **JSON messages**
  **HTTP:** sent as HTTP POST
  **CCS:** encapsulated in XMPP messages.

Before we go on with implementing GCM Server, it is important to pay attention to types of messages. Every message sent in GCM has payload limit of 4096 bytes and by default it is stored by GCM server for 4 weeks. There is two main types "send-to-sync" (collapsible message) and "message with payload" (non-collapsible message). Send-to-sync message tells a mobile device to sync data from server, where the most recent message is relevant. New message will replace the preceding message.

GCM message parameter `collapse_key` plays here an important role because it's used to collapse a group of similar messages when the device is offline. This means that only the most recent message gets sent to the client app. GCM allows a maximum of 4 different collapse keys to be used by GCM server at given time. More than four collapse keys could cause unpredictable consequences.

Messages with payload are delivered every time; `collapse_key` is omitted; the payload the message can be up to 4kb and after 100 stored messages, GCM will discard old messages. For demonstration purposes, we will not use our own 3-rd-party server implementation, because we can use "*Push Bot*" services API for push messing described in next chapter. Push Bot can push notifications via XMPP to feeds that support server-to-server PuBSUBHubbub *protocol*. These Servers can get near-instant notifications when a topic they're interested in is updated (PushBots, 2012). To use this API, in the Android project "*Pushbots131beta.jar*" library should be included. With this library, inside the onCreate(Bundle bundle) method, developers can implement push notification for mobile devices with a one single line of code:

```
Pushbots.init(this, "Sender_ID",
                          "Pushbots_App_ID");
```

*Code 6 – PushBot function*

When the message is sent using the GCM on the screen of android Mobile device, there will be the following content in the notification area:


*Figure 5 – Message sent to device via 3rd party application server*

### 3.3 User notification

User notification is feature that enables 3rd-pary app server to send a single or multiple messages to client apps and app instance that a user owns to reflect the latest messaging state. The way this works is that during registration, the 3-rd party server requests a notification key which is used as an address to send messages (Android Developers, 2014a). The notification key maps a particular use to all of the user's associated registration IDs, allowing him to send message to all of the user's regIDs. To create a user notification key, a JSON request must be sent to GCM Notification endpoint. The notification key for a user's device is only stored on the server. This feature can be used with either the XMPP or HTTP connection server. The Figure shows how communication works with CCS console:

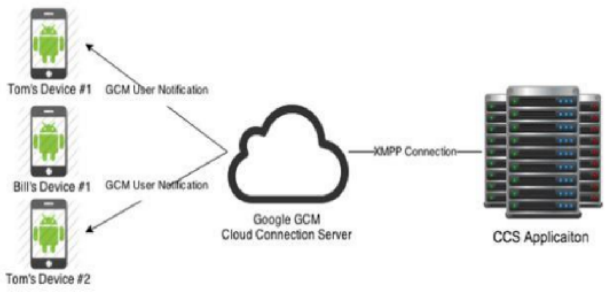


*Figure 6 – GCM User Notification using CCS (Cambell, 2013)*

To use these feature developers can perform generate/add/remove operations, and send upstream messages. To send a message, the application server issues a POST request to URL `https://android.googleapis.com/gcm/notification`. And the following JSON format will be used:

```
//header, project ID API key for
                              authentication
content-type: "application/json"
Header : "project_id": <projectID>
Header: "Authorization", "key=API_KEY"
Request:
{
"operation":["create","add","remove"],
"notification_key_name":"user1_hash",
"registration_ids":["XYZ", "ABC"]
 }
```

*Code 7 – Request Format to work with notification key operations*

Here is a code example that shows targeting a notification key for sending the message, which Bundle data consists of a key/value pair:

```
GoogleCloudMessaging gcm =
GoogleCloudMessaging.get(context);
String to = NOTIFICATION_KEY;
AtomicInteger msgId = new
AtomicInteger();
Bundle data = new Bundle();
data.putString(„26", „case26");
gcm.send(to, id.toString(), data);
```

*Code 8 – Upstream messaging*

## 4. VULNERABILITIES AND THREADS

Last year at the end of July, Kaspersky Lab, computer Security Company, has announced that they have found a backdoor in GCM service. Backdoor is defined as method of bypassing normal access to a program, online service or entire computer system, securing illegal remote access to a computer, while attempting to remain undetected. In our case, the backdoor can be used by criminals to send SMS messages to premium-rate numbers, for stealing data from Android devices and also to send messages which content includes links to itself or other malware. Previously mentioned, messages are sent in JSON format what enables hackers to take advantage for malicious purposes. Once gained a GCM ID, malware updates are distributed exploiting directly the GCM services. Also the Command to the malicious agent is sent is by exploiting the service and using JSON format. Using GCM as Command and Control server for Android Malware is already generally known concept. The top three most Android Trojans that used JSON format are listed below:

- SMS.AndroidOS.OpFake.bo
- SMS.AndroidOS.FakeInst.a
- SMS.AndroidOS.OpFake.a

*"Trojan-SMS.AndroidOS.OfFake.bo is one of the most sophisticated SMS Trojans. Its distinguishing features are a well-designed interface and the greed of its developers. When launched, it steals money from the mobile device's owner – from $9 to the entire amount in the user's account. There is also the risk of the user's telephone number being discredited, since the Trojan can collect numbers from the contact list and send SMS messages to all of those numbers. The malware targets primarily Russian-speakers and users in CIS countries."* (Chebyshev and Unuchek, 2013)

Kaspersky Lab has reported Google about the discovered vulnerabilities, and announced that the only way to protect users against such attacks is through blocking developers' accounts with IDs linked to the registration of malicious applications. The number of malware that exploits the GCM service is relatively low. Most malware are prevalent in Western Europe (particularly in Russia) and Asia.

## 5. CONLUSION

This paper briefly introduced Google Cloud Messaging for Android (GCM) which is defined as service that allows developers to send data from 3rd party server to their users' Android-powered devices, and also to receive messages from devices on the same connection. The GCM service handles all aspects of queuing and delivery to the target Android application. Key concepts of GCM are the **components** which make the architecture of GCM and the **credentials** used for access authentication in different stages of GCM. *Components* include client app that runs on Android device, 3rd-party app server and GCM connection server. *Credentials* are the IDs and tokens used in different stages of GCM (Sender ID, application ID, registration ID, google user account and sender auth token). GCM has many useful features such as: "Send to Sync" and "Send data" capabilities, support for multicast messages, delay while idle, and the use of persistent connections (by using XMPP), the upstream messaging and synchronization notification across multiple android devices etc. The new CCS API for GCM messages has made sending messages faster than the HTTP API, but with the added complexity of an asynchronous process. GCM is a great way for Android developers to manage the communication with their applications and users, but it also has important disadvantages that should be taken in consideration.

**References:**


1 Android Developers, 2014a. GCM Overview [WWW Document]. Google Cloud Messaging - Overview. URL http://developer.android.com/google/gcm/gcm.html (accessed 5.9.14).

2 Android Developers, 2014b. Implementing GCM Client [WWW Document]. Implementing GCM Client. URL http://developer.android.com/google/gcm/client.html (accessed 5.9.14).

3 Android Developers, 2014c. Implementing GCM Server [WWW Document]. Implementing GCM Server. URL http://developer.android.com/google/gcm/server.html#choose (accessed 5.12.14).

4 Cambell, A., 2013. Google Cloud Messaging: Cloud Connection Server Tutorial [WWW Document]. CapTech consulting. URL http://captechconsulting.com/blog/antoine-campbell/google-cloud-messaging-cloud-connection-server-tutorial (accessed 5.9.14).

5 Chebyshev, V., Unuchek, R., 2013. Mobile Malware Evolution: 2013 - Securelist [WWW Document]. URL http://www.securelist.com/en/analysis/204792326/Mobile_Malware_Evolution_2013?print_mode=1 (accessed 5.19.14).

6 Freelancersnepal, 2014. Google Cloud Messaging (GCM) in Android using PHP Server [WWW Document]. Freelancersnepal. URL http://freelancersnepal.wordpress.com/2014/01/29/google-cloud-messaging-gcm-in-android-using-php-server/ (accessed 5.9.14).

7 Google Developers, 2013. Google I/O 2013 [WWW Document]. Google I/O 2013. URL https://developers.google.com/events/io/ (accessed 5.26.14).

8 HMKCode, 2014. Android Google Cloud Messaging Tutorial [WWW Document]. Android Google Cloud Messaging Tutorial. URL http://hmkcode.com/android-google-cloud-messaging-tutorial/ (accessed 5.9.14).

9 PushBots, 2012. PushBots: communicate with your mobile app users in minutes [WWW Document]. PushBots. URL https://pushbots.com (accessed 5.26.14).

## Information on authors:

**Zoran Kos B.Sc**

e-mail: zoran.kos@foi.hr

Zoran Kos is 2nd year student of Information and Programming engineering of Faculty of Organization and Informatics in Varaždin. Main interests are programming web, desktop and mobile application, new IT trends like cloud computing and mobility, making and presenting business models for startup projects. While developing new apps he focused on front-end development. He already has few smaller but successful projects behind him.

Zoran is the main contributor to this paper. He wrote the first version of the text and did coding of the presented examples.

**Zlatko Stapić, PhD.**

e-mail: zlatko.stapic@foi.hr

Faculty of Organization and Informatics

Pavlinska 2, 42000 Varaždin, Tel: +385 42 390 820, Fax: +385 42 213 413

Zlatko Stapić, PhD., works at the Information Systems Development Department at Faculty of Organization and Informatics in Varaždin. He obtained his PhD in computer sciences from University of Alcalá (Spain) and in information sciences from University of Zagreb (Croatia) in cotutelled doctorate program. His scientific and research interests include software- and mobile applications development methodologies. He participated in more than 15 scientific and professional projects and published more than 30 scientific and professional papers. Currently he leads the *Laboratory for Development and Transfer of Mobile Technologies* (FOI MT-Lab). Zlatko is putting a special focus in inclusion of students in his scientific and professional activities. The published papers, projects, awards and other relevant information can be found on his personal website: http://www.foi.unizg.hr/djelatnici/zlatko.stapic.

Zlatko's contributions to this paper are only of corrective nature. He mentored student Zoran Kos in defining the body, the structure and the style of the paper. After the first version of the paper is written, he proof-read the text and made necessary changes to make the text more focused and readable.