Ottimizzazione delle risorse nell'uso di servizi in background in SeismoCloud per Android

Enrico Bassetti

Dipartimento di Informatica
Università La Sapienza - Roma

Dicembre 2017

Le motivazioni alla base di questo lavoro

In uno *smartphone*, come in molti ambienti embedded, le risorse sono limitate. Utilizzarle correttamente non è solo un esercizio di stile o una azione eticamente corretta (nei confronti dell'utente, a cui le risorse "appartengono"), ma anche un modo per rendere il sistema efficiente.

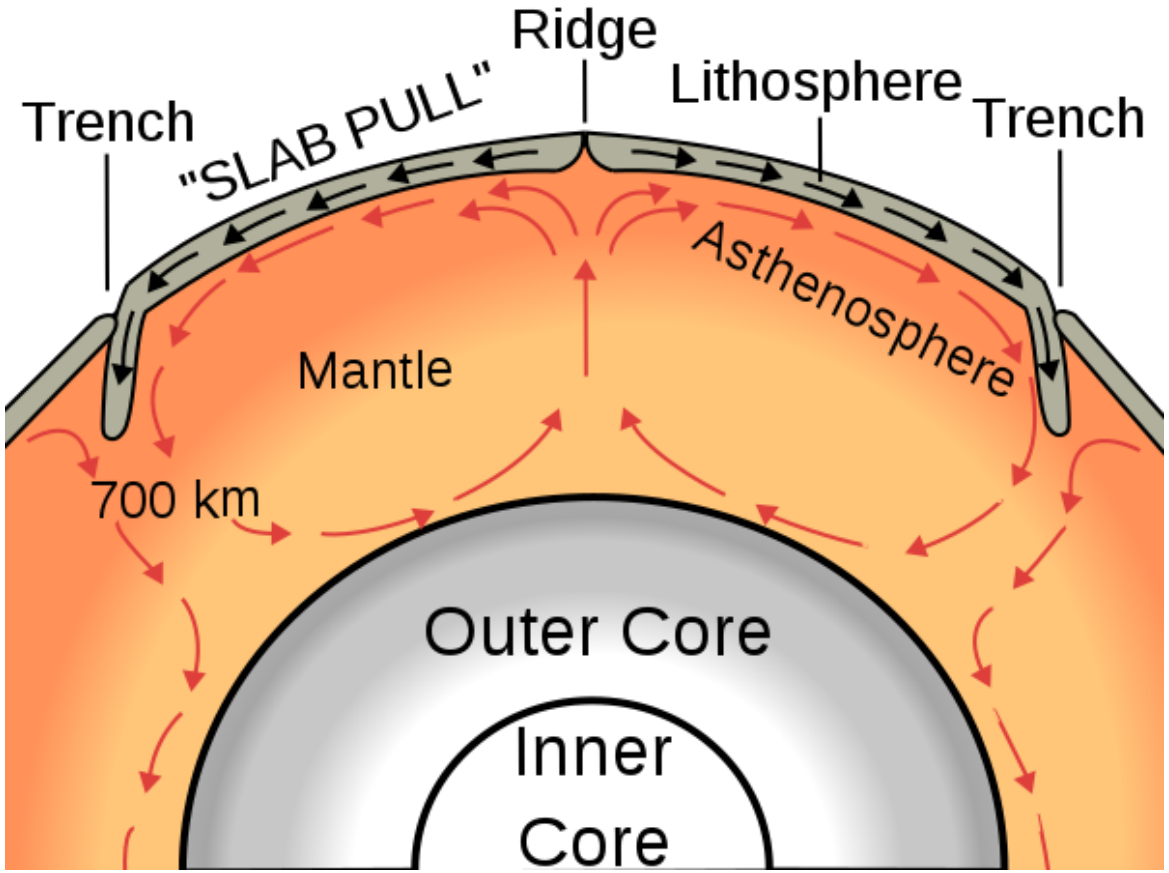
Un sistema che è in grado di fornire una migliore *user experience* (grazie all'efficienza) gratifica l'utente nella scelta della applicazione.

Roadmap

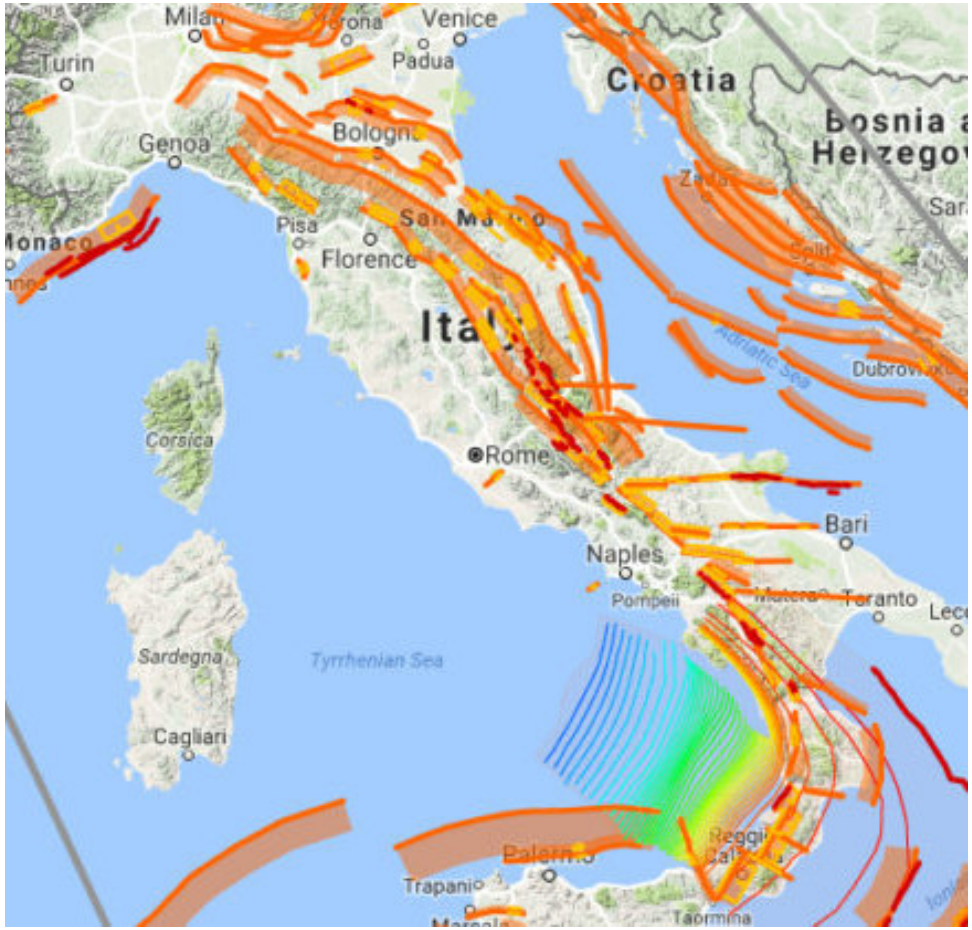
- I Terremoti
- Il progetto SeismoCloud
- Ottimizzazione energetica
- Ottimizzazione della rete
- Ottimizzazione in fase di sviluppo
- Sviluppi futuri

I terremoti

Origine


Ridge
Lithosphere
Trench
Trench
"SLAB PULL"
Asthenosphere
Mantle
700 km
Outer Core
Inner
Core

Le faglie italiane



Previsione, prevenzione e gestione

Nonostante i progressi nella ricerca, **i terremoti non sono (ancora) previdibili**.

Dobbiamo quindi affidarci alla **prevenzione** e alla **gestione dell'emergenza**.

Mentre la rete IV dell'Istituto Nazionale di Geofisica e Vulcanologia è utile in fase di prevenzione (grazie ai dati accumulati e studiati nel tempo), il progetto **SeismoCloud** si posiziona tra il momento in cui il terremoto si presenta nell'epicentro, ed il momento in cui colpisce le zone circostanti.

SeismoCloud

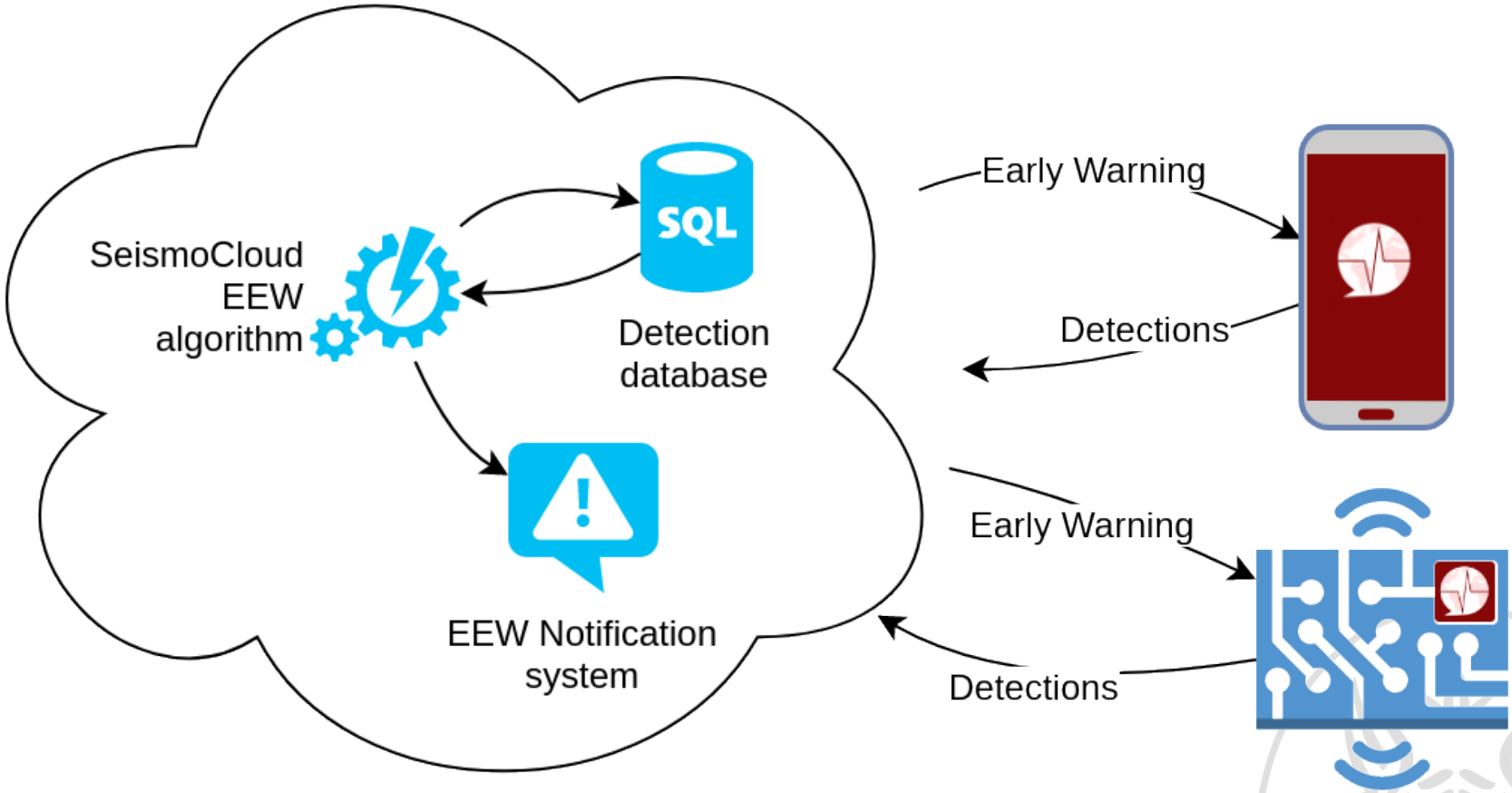
SeismoCloud

SeismoCloud è un progetto di *early warning* mediante *crowdsourcing*.

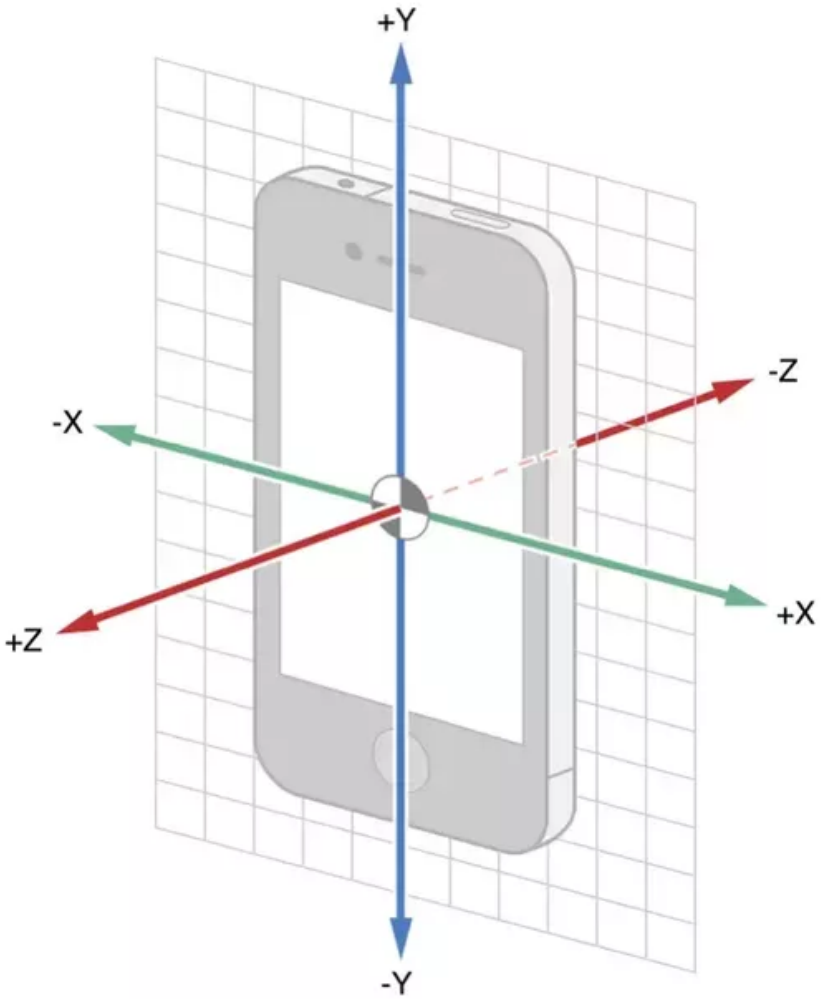
Grazie ad una rete di sensori fissi e mobili, il sistema acquisisce i dati sulle possibili scosse, li analizza ed eventualmente genera una notifica.

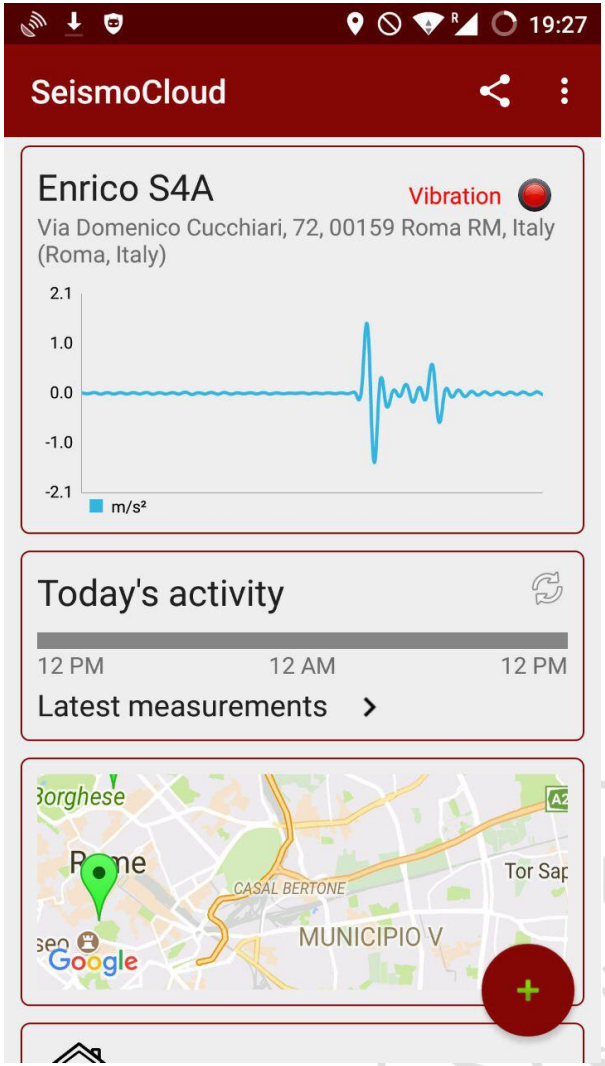
L'obiettivo di SeismoCloud è quello di **anticipare la scossa di terremoto**, grazie alla differente velocità delle onde radio e delle scosse di terremoto.

Architettura



I telefoni come sensori





Ottimizzazione energetica

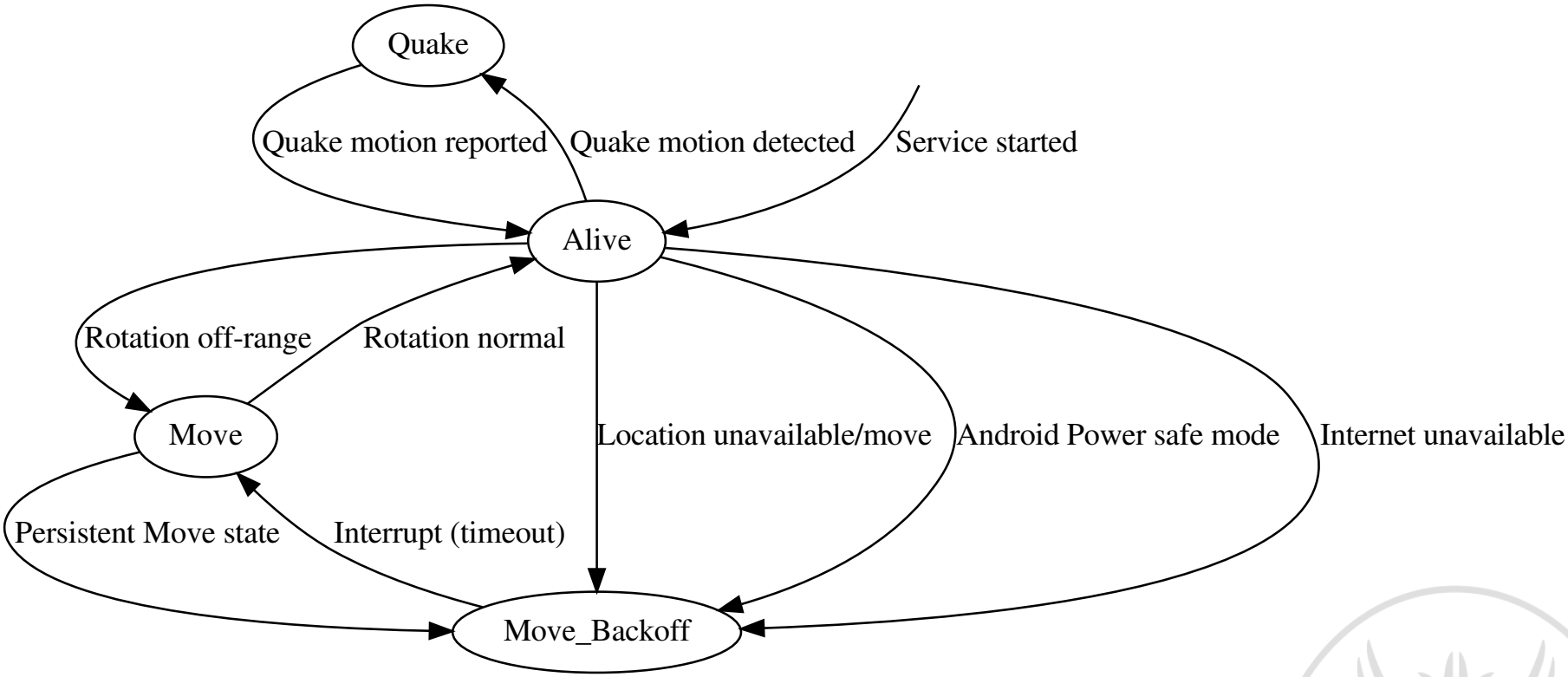
Contesti di attivazione (del servizio in background)

Un **contesto di attivazione** è un insieme di variabili che, per opportuni valori, consentono al servizio di attivarsi.

In **SeismoCloud**, il contesto è definito come:

- Rotazione/posizione del telefono
- Presenza o meno della localizzazione (GPS)
- Presenza o meno di uno spostamento (geografico)
- Presenza o meno della rete Internet
- Capacità rimanente batteria

Contesti di attivazione: diagramma di stato

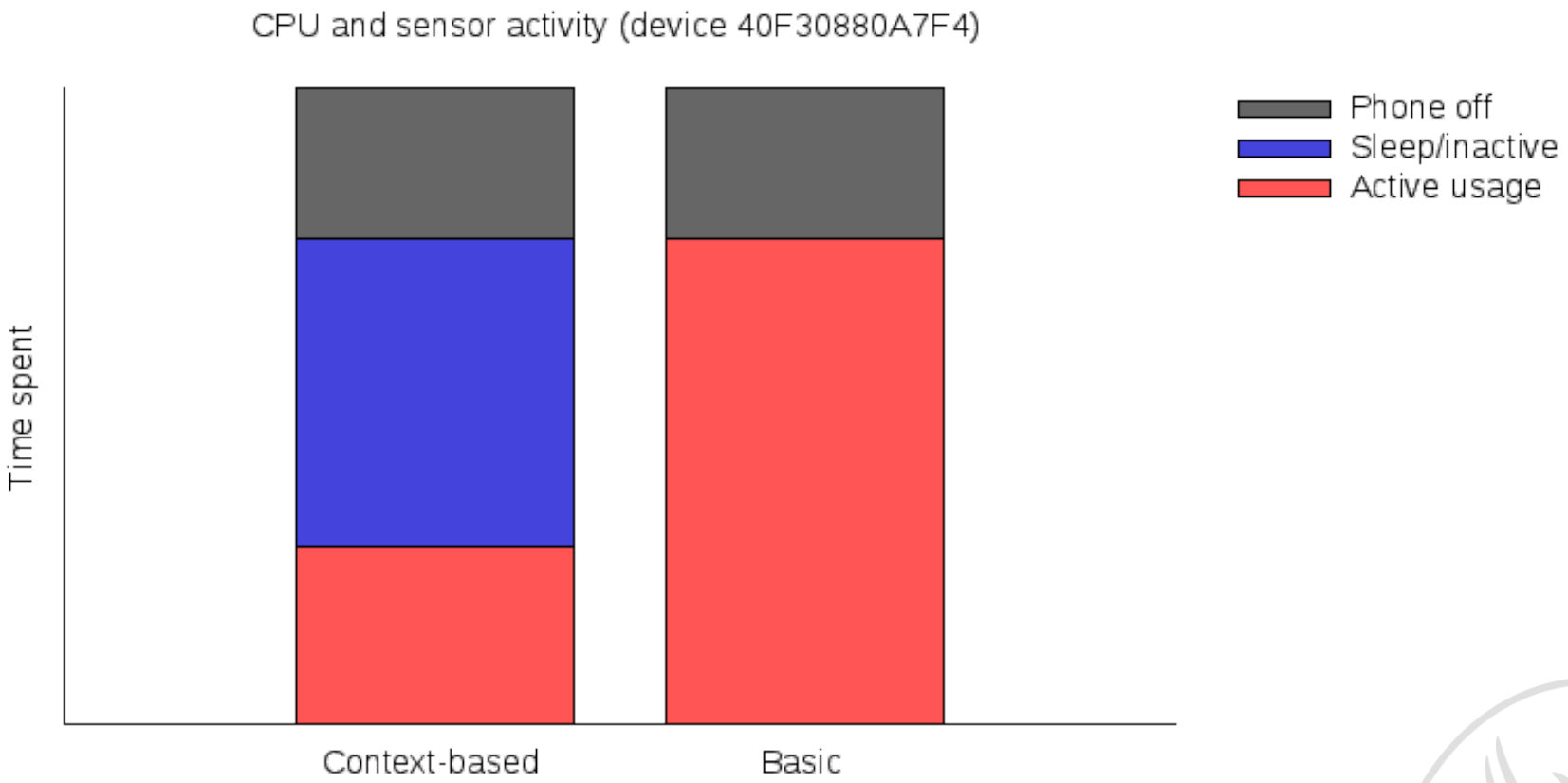


Contesti di attivazione: risultati

	Algoritmo originale	**Algoritmo ottimizzato**
Servizio attivo	18h 24m	6h 36m
Servizio inattivo	mai	11h 48m
Telefono spento	5h 36m	5h 36m

Tabella: Tempo di utilizzo delle risorse (media giornaliera)

Contesti di attivazione: risultati



I sensori

La app **SeismoCloud**, per effettuare la rilevazione, inizialmente utilizzava solo l'*accelerometro*.

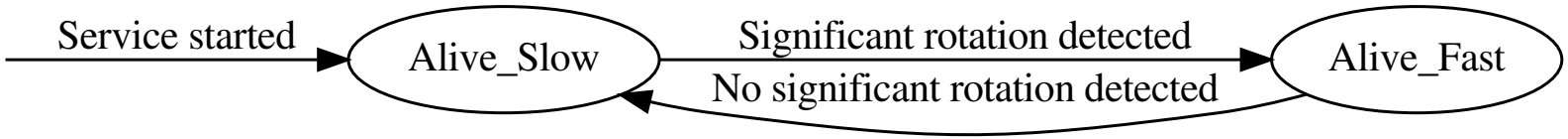
Per l'implementazione dei contesti di attivazione, è necessario utilizzare anche un sensore per la rotazione.

I sensori: giroscopio e magnetometro

- Una prima implementazione prevedeva l'uso del **giroscopio**
- Successivamente, si è affiancato l'uso del **magnetometro** come alternativa per telefoni di fascia bassa
- Il periodo di sampling, per entrambi i sensori (*accelerometro* e *giroscopio/magnetometro*) era di $20ms$

I sensori: tentativi di ottimizzazione

Una prima ottimizzazione prevedeva l'uso di due periodi di sampling differenti: **ALIVE_FAST** e **ALIVE_SLOW**



Tuttavia, lo studio ha evidenziato che il carico di CPU era equivalente, dunque questa ottimizzazione è stata abbadonata.

I sensori: il consumo

Successivamente, uno studio sul consumo dei sensori ha evidenziato che il consumo del *giroscopio* è di un ordine di grandezza superiore rispetto al consumo dell'*accelerometro+magnetometro* (quasi sempre in un unico integrato).

	In fase di misura	**Stand-by**
Accelerometro + Magnetometro		
ADXL345	$40\mu A$	$0.1\mu A$
Giroscopio		
L3G4200D	$6.1mA$	$0.2mA$

Dunque, è stato scelto di utilizzare sempre e solo **magnetometro** per la rotazione.

La curva di scarica

Gli smartphones moderni hanno una autonomia limitata, dovuta anche (ma non solo) al numero di applicazioni in esecuzione.

Un modo aggressivo per limitare l'uso della batteria da parte di SeismoCloud è l'uso di una **curva di scarica**.

La curva di scarica: definizione

$$\gamma = \frac{85}{720} \tag{1}$$

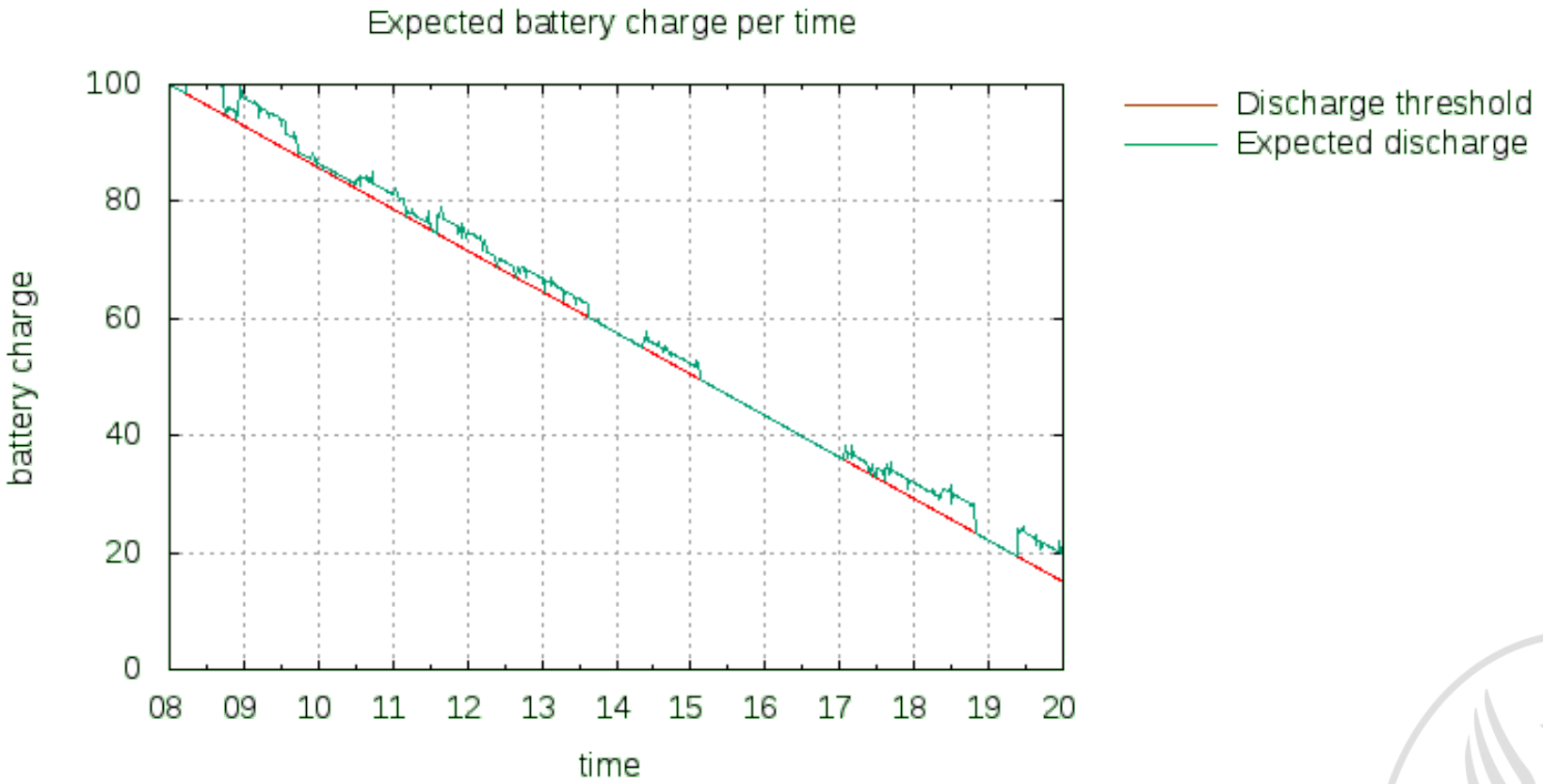
$$\lambda = \text{hour} * 60 + \text{minutes} \tag{2}$$

$$\mathit{Threshold} = \begin{cases} 100 & \text{if hour} < 8 \\ 15 & \text{if hour} > 20 \\ 100 - ((\lambda - 480) * \gamma) & \text{otherwise.} \end{cases} \tag{3}$$

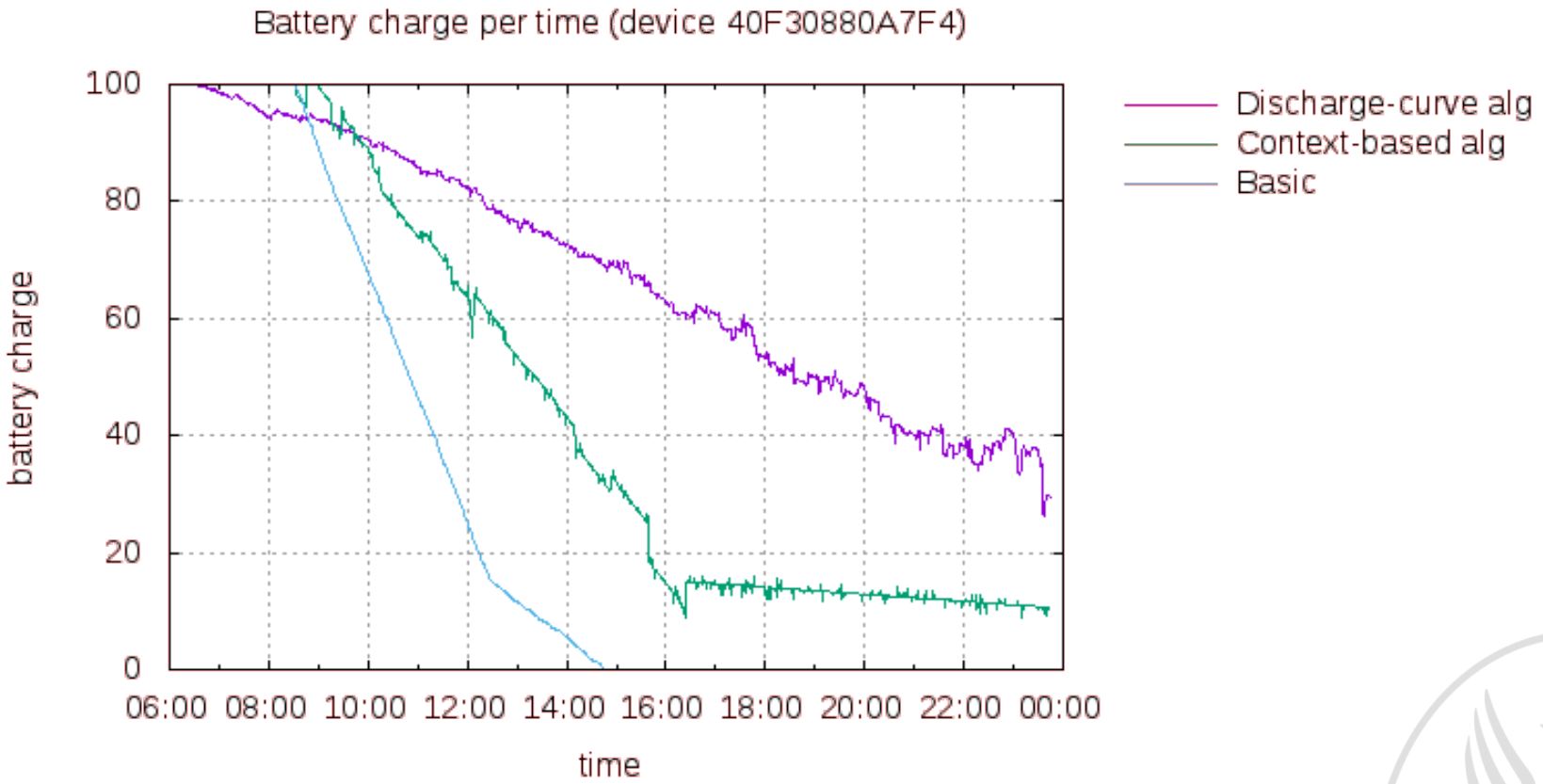
Dove:

- γ: *rate* di perdita capacità batteria per minuto (fascia 8-20)
- λ: numero di minuti da mezzanotte del giorno attuale
- *Threshold*: capacità della batteria (%) prevista per l'orario attuale

La curva di scarica: valore atteso



La curva di scarica: risultati



Ottimizzazione della rete

Uso efficiente della rete

Molti *smartphones* sono connessi alla rete mediante una connessione tariffata *a consumo*. E' quindi importante rimuovere informazioni superflue dalla trasmissione.

Inoltre, un *payload* alto nella trasmissione si traduce in un maggiore consumo di batteria.

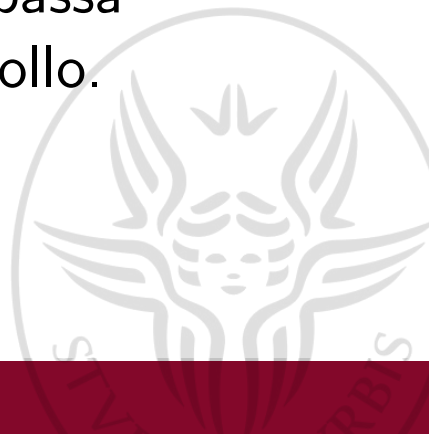
E' quindi importante ottimizzare il protocollo di trasmissione allo scopo di ridurre il carico nella rete.

Message Queue Telemetry Transport

Message Queue Telemetry Transport è un protocollo (standard ISO) di tipo *publish-subscriber*, ottimizzato per operazioni in ambienti con memoria/rete limitati (ad esempio, per l'*Internet of Things*).

Come altri protocolli *publish-subscribe*, è basato sull'invio di messaggi da parte di *publisher*, ruotati verso i sottoscrittori tramite *topic* associati ai messaggio.

L'ottimizzazione, rispetto ad altri protocolli, è data dalla bassa complessità (e limitato numero di funzionalità) del protocollo.



MQTT in SeismoCloud

Prima dell'ottimizzazione, il backend di SeismoCloud forniva solo API in HTTP.

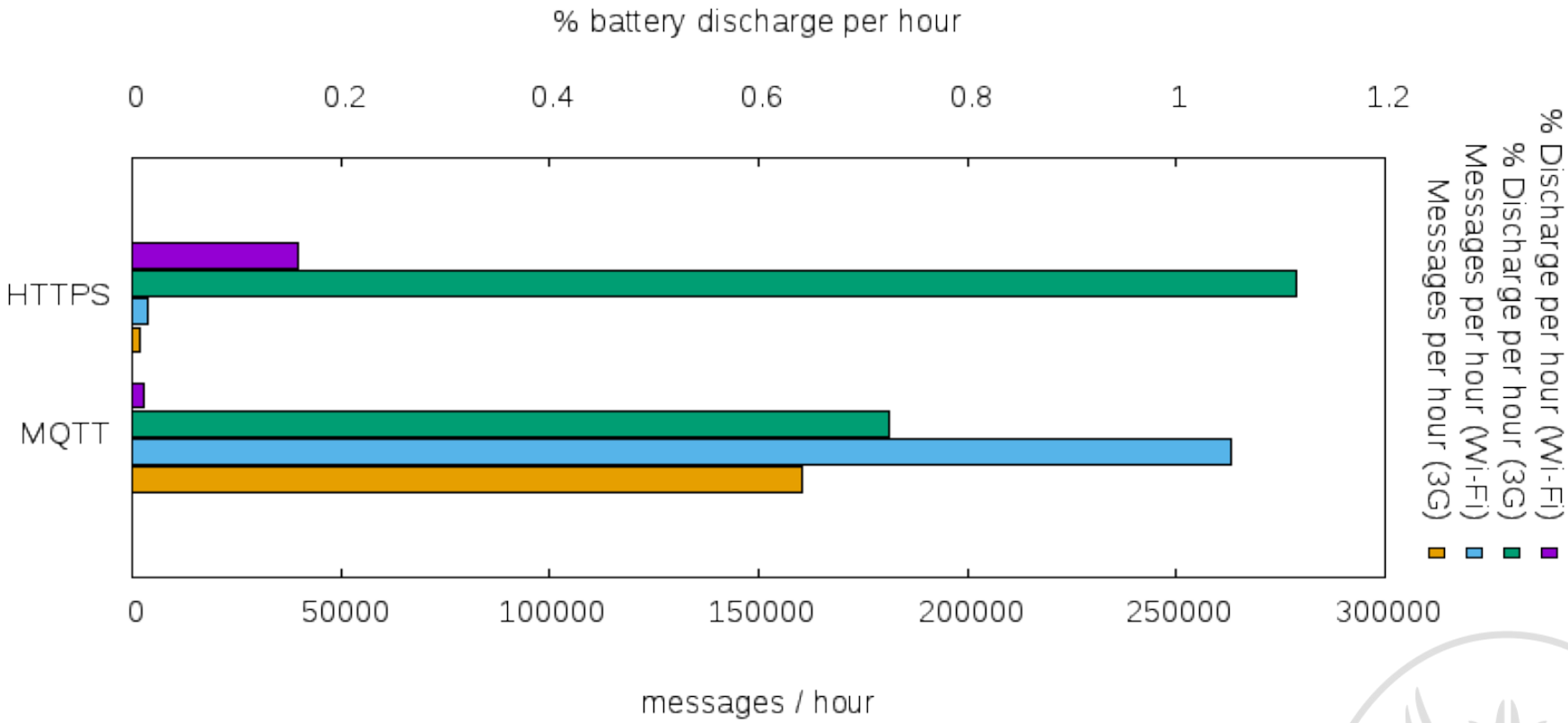
Sebbene le API in HTTP siano ancora presenti (e utilizzate come *fallback*), il backend è stato espanso per supportare MQTT.

L'utilizzo di MQTT, al posto di HTTP, permette alla *App* di avere un basso *footprint*, sia per quanto riguarda il codice in esecuzione, sia per quanto riguarda il traffico di rete.

MQTT vs HTTP

	HTTP	**MQTT**
Overhead	da 200 a 2048 bytes	5-7 bytes
Connessione	1 per richiesta	persistente
Codifica header	RFC 2822	*Type-Length-Value*
Schema	*request/response*	*publisher/subscriber*
QoS	-	3 Livelli
Dim. libreria	Apache HTTP: 3 MB	Eclipse Paho: 14.6 KB

MQTT vs HTTP


% battery discharge per hour
0
0.2
0.4
0.6
0.8
1
1.2
HTTPS
MQTT
0
50000
100000
150000
200000
250000
300000
messages / hour
% Discharge per hour (Wi-Fi)
% Discharge per hour (3G)
Messages per hour (Wi-Fi)
Messages per hour (3G)

Ottimizzazione in fase di sviluppo

Android Annotations: prima

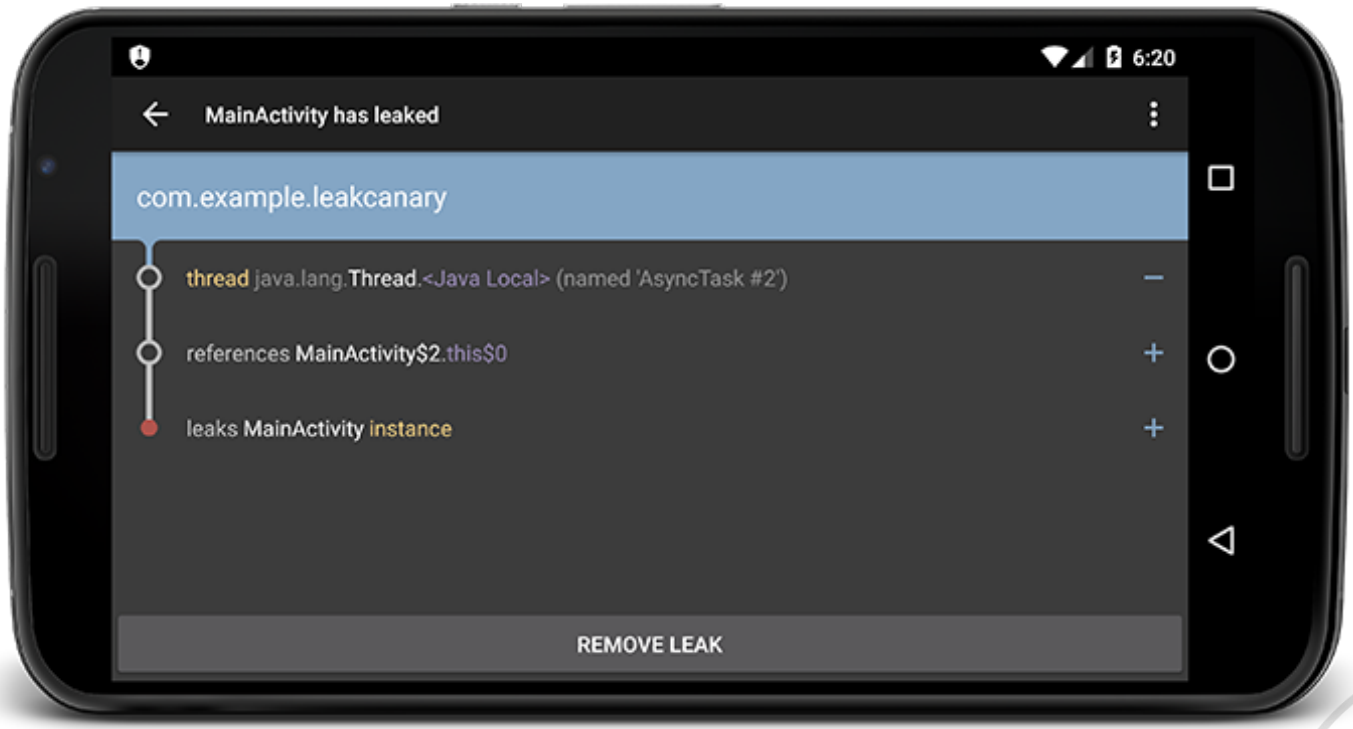
```
1  public class SurveyActivity extends AppCompatActivity {
2          private String param1;
3          private String param2;
4
5          @Override
6          protected void onCreate(Bundle savedInstanceState) {
7                  super.onCreate(savedInstanceState);
8                  setContentView(R.layout.activity_survey);
9
10                 Intent in = getIntent();
11                 if (in == null) {
12                         // Gestione dell'errore
13                         this.finish();
14                 }
15                 param1 = in.getStringExtra("param1");
16                 param2 = in.getStringExtra("param2");
17         }
18         // Altro codice della activity
19 }
```

Android Annotations: dopo

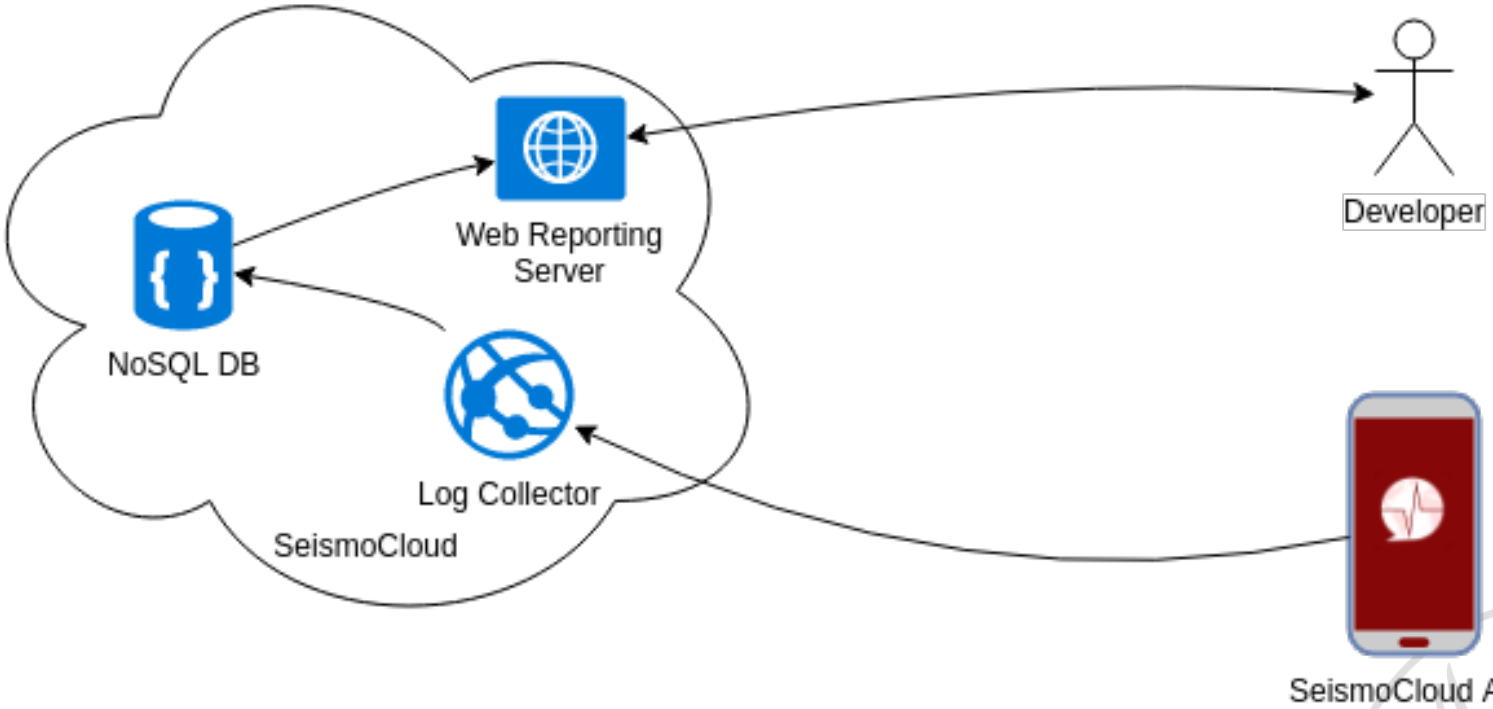
```
1 @EActivity(R.layout.activity_survey)
2 public class SurveyActivity extends AppCompatActivity {
3         @Extra String param1;
4         @Extra String param2;
5         // Altro codice della activity
6 }
```

Grazie ad *Android Annotations* il risparmio medio del codice è del 30%

LeakCanary

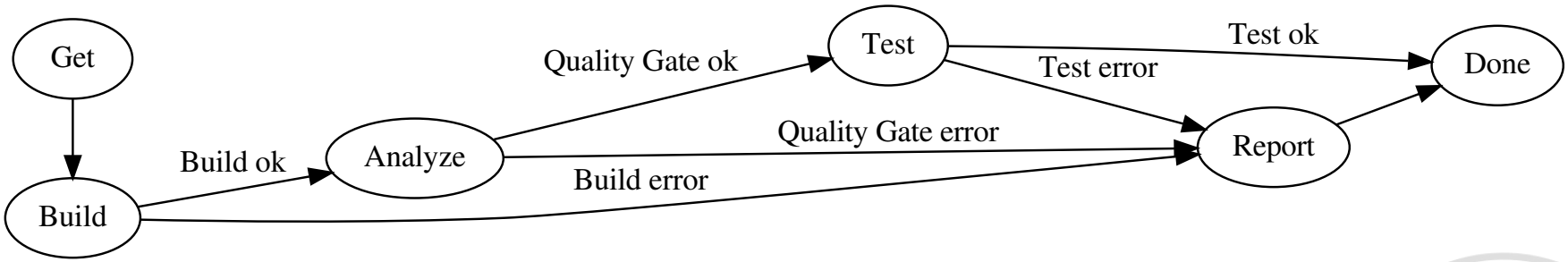

6:20
MainActivity has leaked
com.example.leakcanary
thread java.lang.Thread.<Java Local> (named 'AsyncTask #2')
references MainActivity$2.this$0
leaks MainActivity instance
REMOVE LEAK

Application Crash Reports for Android

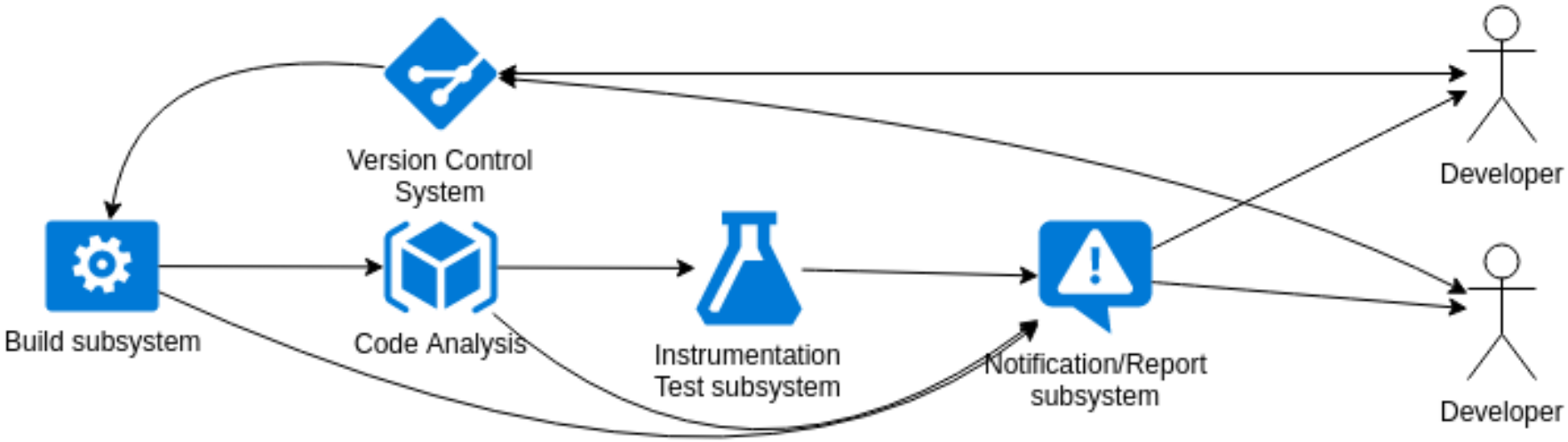


Continuous integration

Per mantenere sempre sotto osservazione il codice, è stata sviluppata una *pipeline* per il processi di build utilizzando **SonarQube**, **Jenkins** e *tool* specifici di Google per Android:



Continuous integration



Sviluppi futuri

Sviluppi futuri

1. Sistema di spegnimento *a zone geografiche*
2. Utilizzo di sensori dedicati (se presenti)
3. Porting del codice di pre-filtro in modo nativo (C/C++) per ottimizzazione energetica
4. Tecniche avanzate per l'ottimizzazione del rilevamento della posizione geografica

Grazie