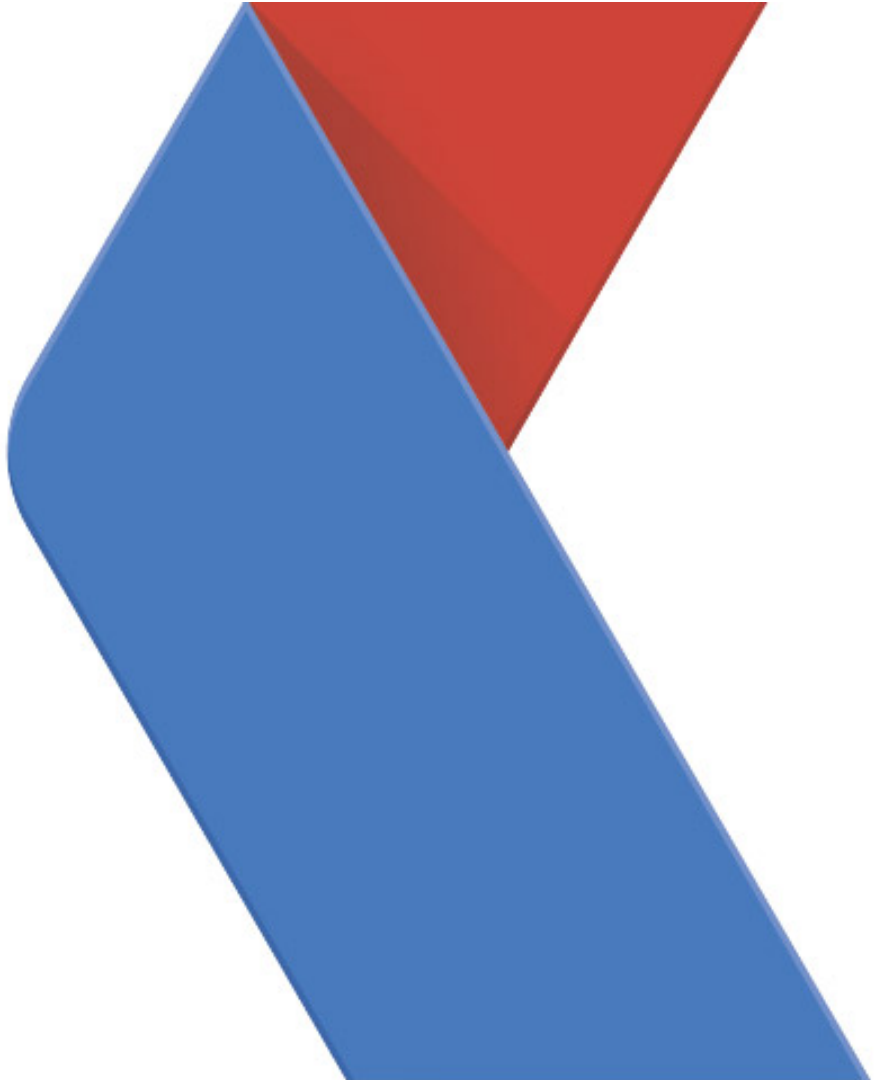
Android
电池和内存优化

December 2016



议程

- 电池优化
- 内存优化
- APIs 以及 诊断工具

“Our One Wish? Longer Battery Life”
-The Wall Street Journal

电池优化

几百个电池应用（Google Play 商店）

10M-100M+ 下载量

http://www.wsj.com/articles/our-one-wish-longer-battery-life-1424650700

电量去了哪?

设计原则：Background Process

减少

*减少所有**background process**。*

延迟

*若一定要跑background process，就应该延迟到设备**充电**时。*

合并

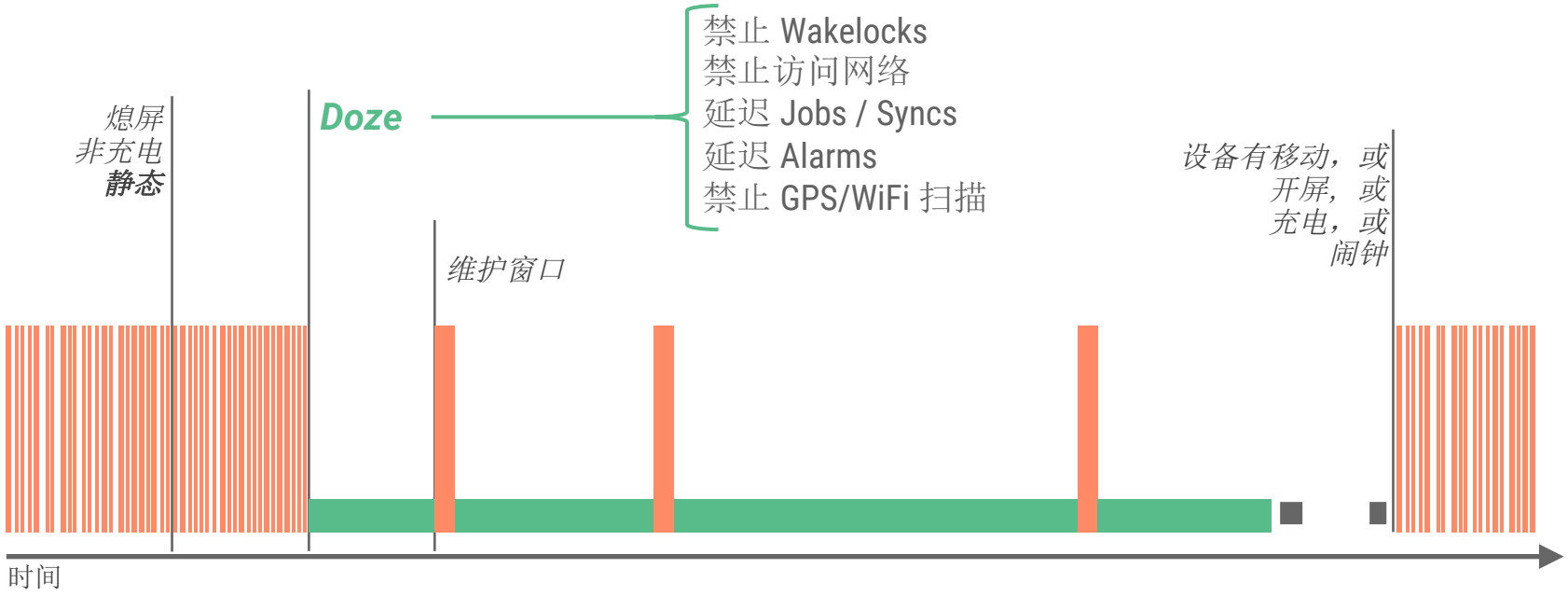
若不能延迟，就应该与其它background process合并，以减少开销成本。

Doze 和 App Standby

Android **Marshmallow**

Doze 时间线 (Android 6.0 / Marshmallow)

合并



Doze 需要 significant motion detector

App Standby

延迟

应用**未使用**一段时间

App Standby

“**使用**”是什么意思？

- App 有 **foreground process** (activity 或 service)
- App 在锁屏上或notification tray上有有**notification**
- 用户明确启动App

当设备在非充电情况下时：

- App **不能访问网络**
- App 的 **sync／jobs 受到延迟**

针对 Doze 和 App Standby 做优化

- 高优先级（**High-priority**）的 GCM (FCM) messages
  - 容许 apps 短暂的获取 wakelock，访问网络
  - 适用于即刻通知用户的场景
  - 非优先级的 FCM messages 延迟到维护窗口

- Doze 和 App Standby 在AOSP是默认被**disabled**
  - 需要 Firebase Cloud Messaging (Google Play Services) 或其他云端推送服务

http://developer.android.com/training/monitoring-device-state/doze-standby.html

针对 Doze 和 App Standby 做优化

- **Foreground** services
  - 不受Doze 和 AppStandby 影响
  - 适用于播放音乐和其它类似的用户场景
- **Alarm** APIs
  - 适用于闹钟等场景
- 白名单
  - 用户可以手动把 app 加到白名单
  - Apps 可以在运行时要用户加到白名单，但**必须满足可接受的用例要求**，并且通过 **Play Store review**

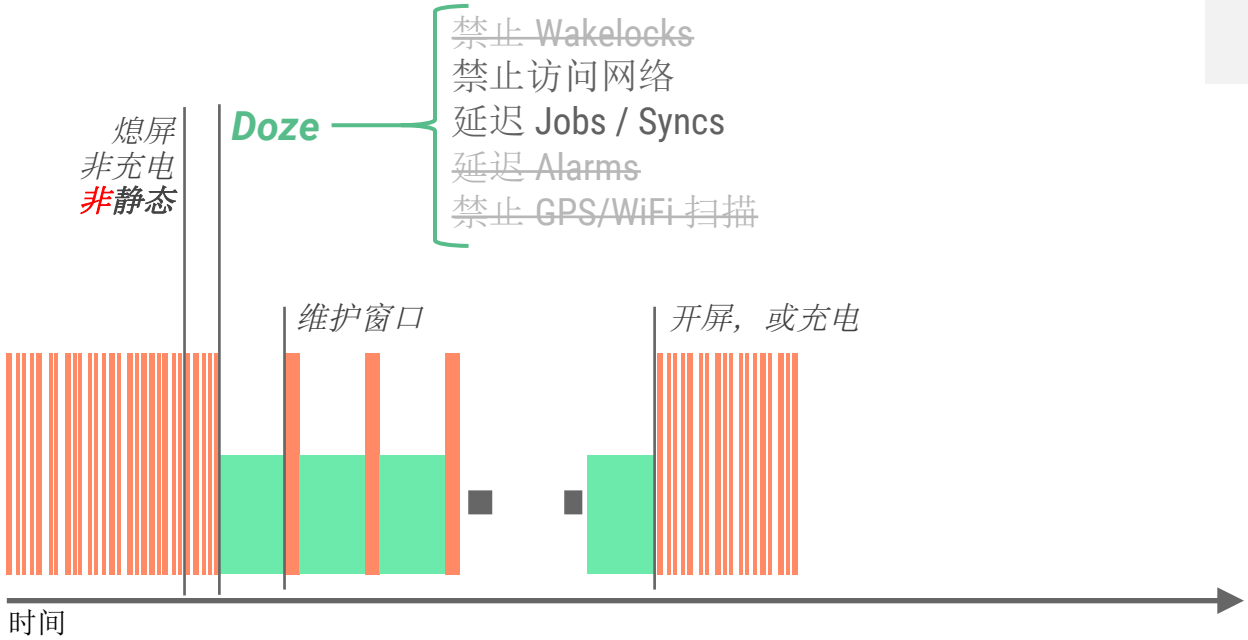
http://developer.android.com/training/monitoring-device-state/doze-standby.html

Doze

Android **Nougat**

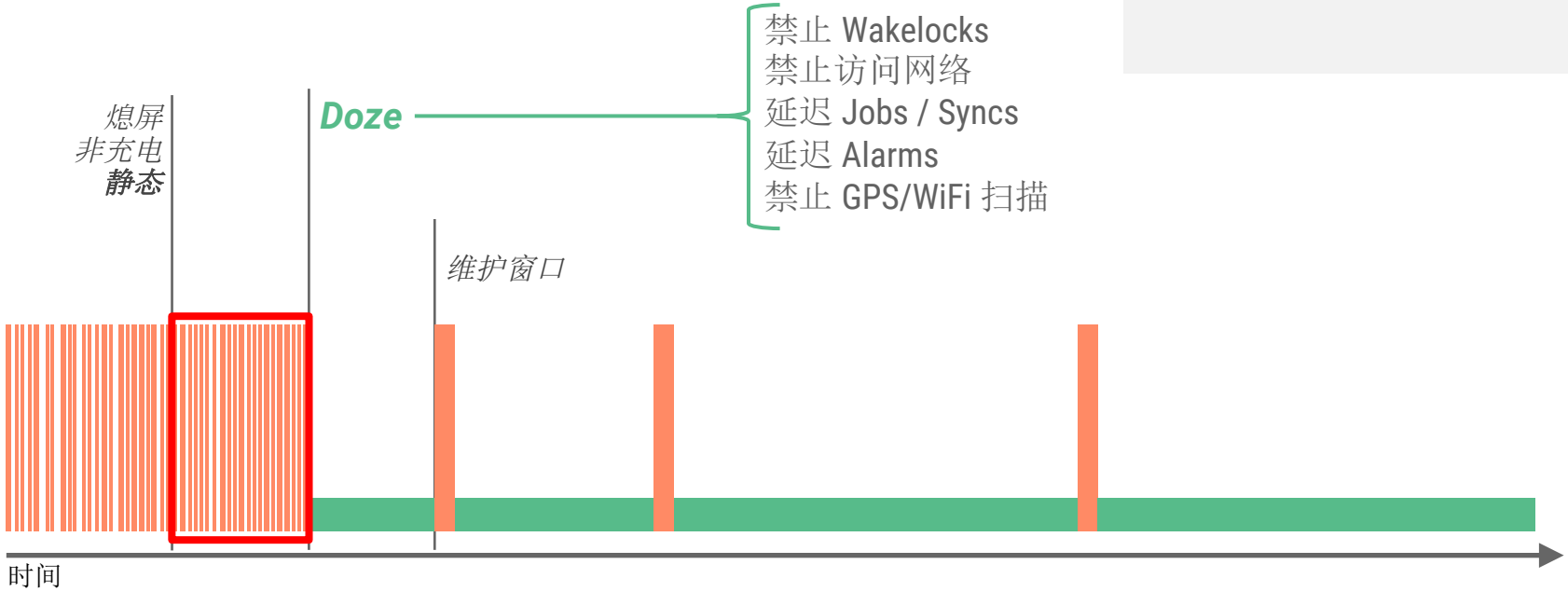
Doze 时间线 (Android 7.0 / Nougat)

合并



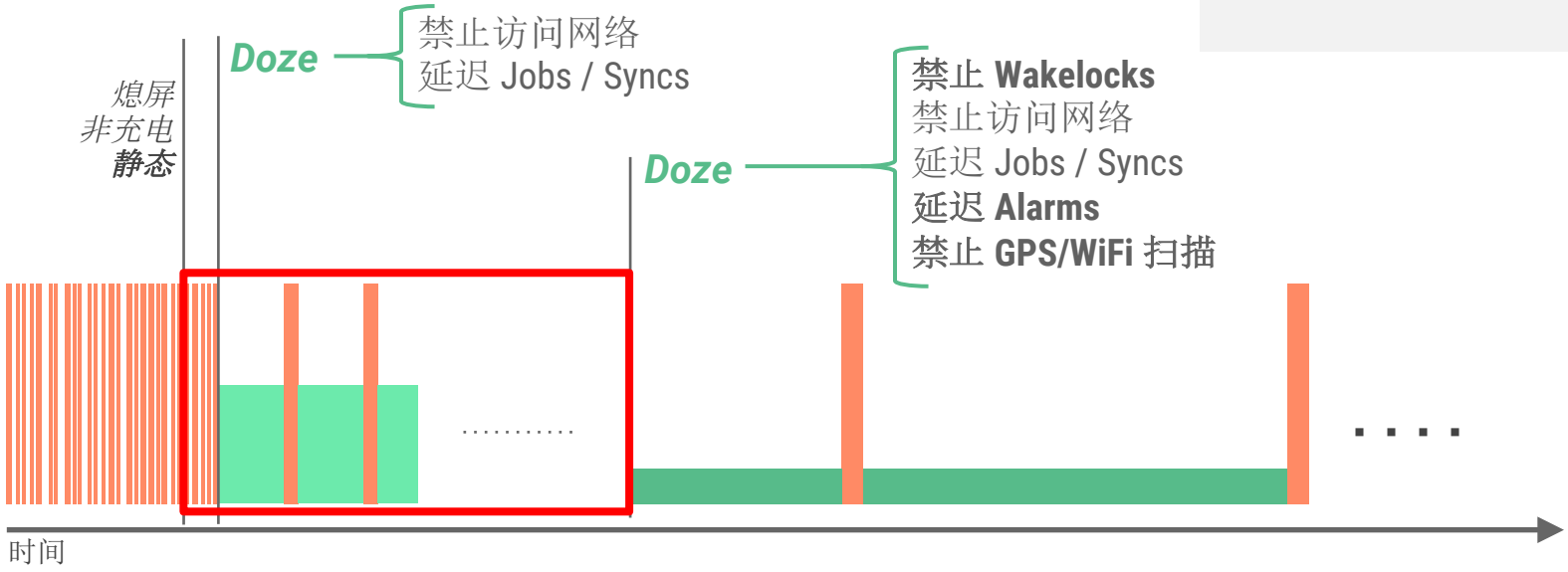
重温: Doze timeline (Marshmallow)

合并



Doze 时间线 (Android 7.0 / Nougat)

合并



针对 Doze 和 App Standby 做优化

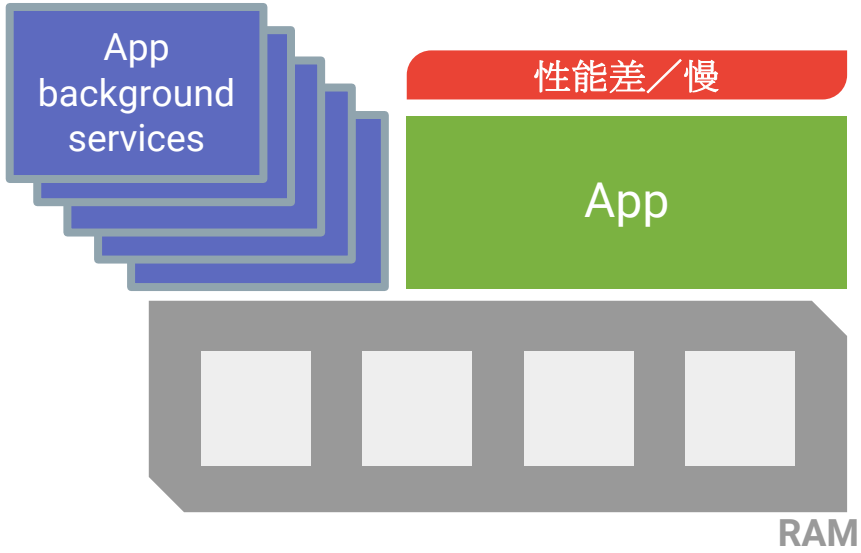
- 高优先级（**High-priority**）的 GCM (FCM) messages
  - 容许 apps 短暂的获取 wakelock，访问网络
  - 适用于即刻通知用户的场景

- **Foreground** services
  - 不受Doze 和 AppStandby 影响
  - 适用于播放音乐和其它类似的用户场景

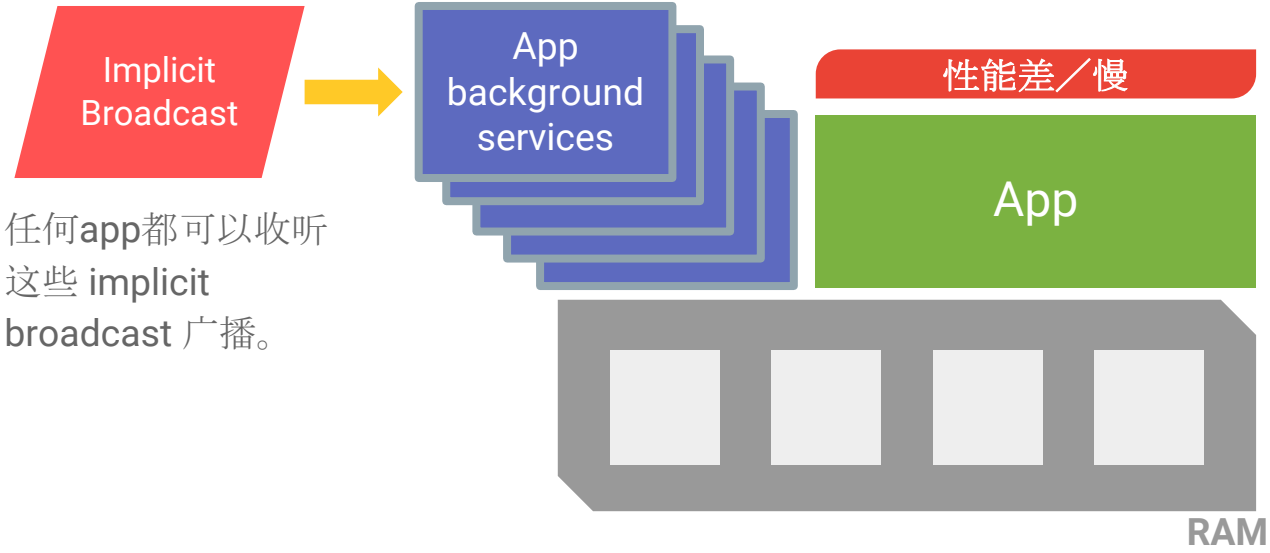
http://developer.android.com/training/monitoring-device-state/doze-standby.html

内存优化

*低内存设备*上的内存抖动



*低内存设备*上的内存抖动

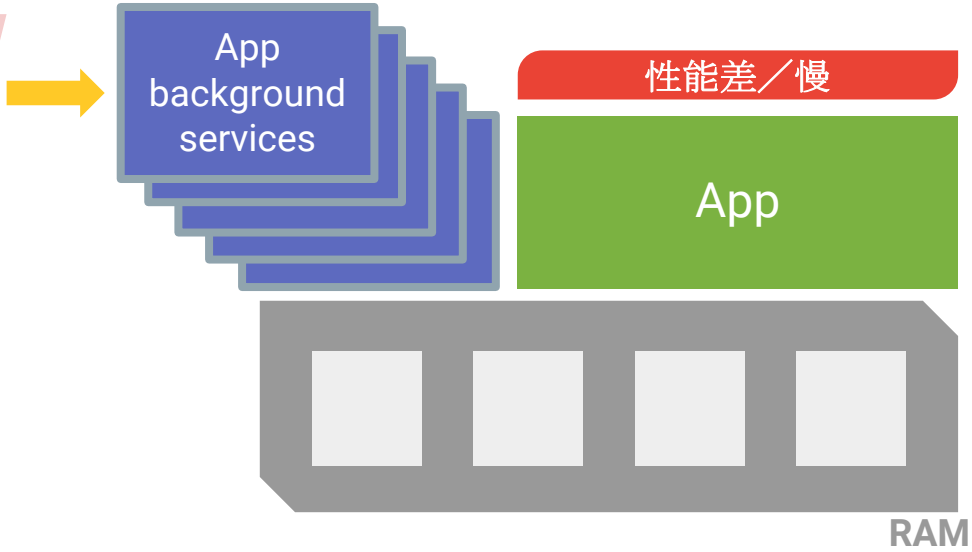


任何app都可以收听这些 implicit broadcast 广播。

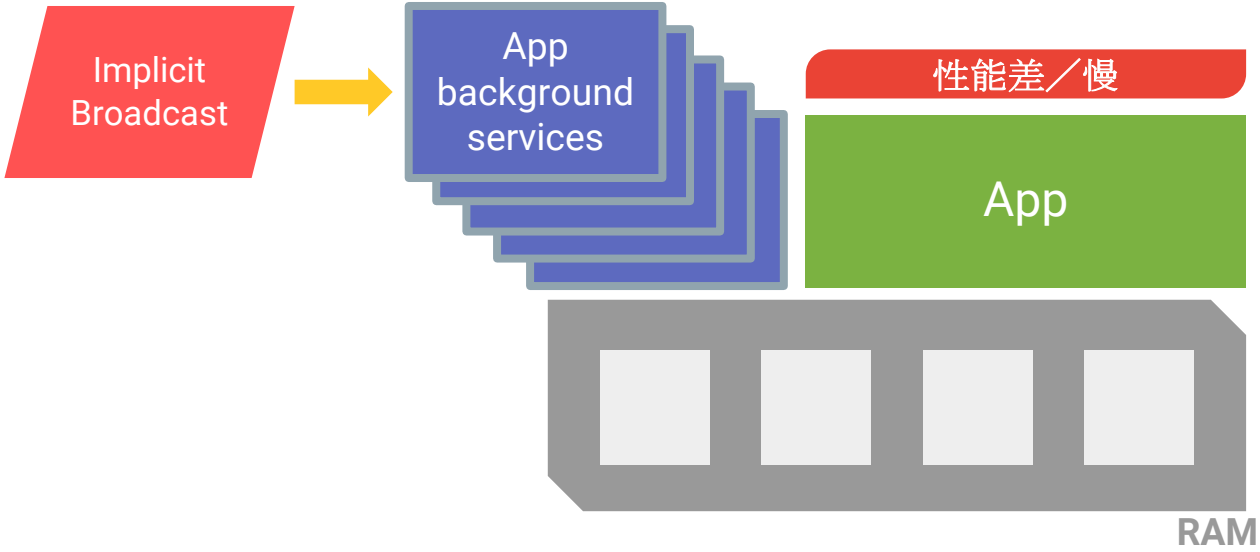
*低内存设备*上的内存抖动

CONNECTIVITY_CHANGE

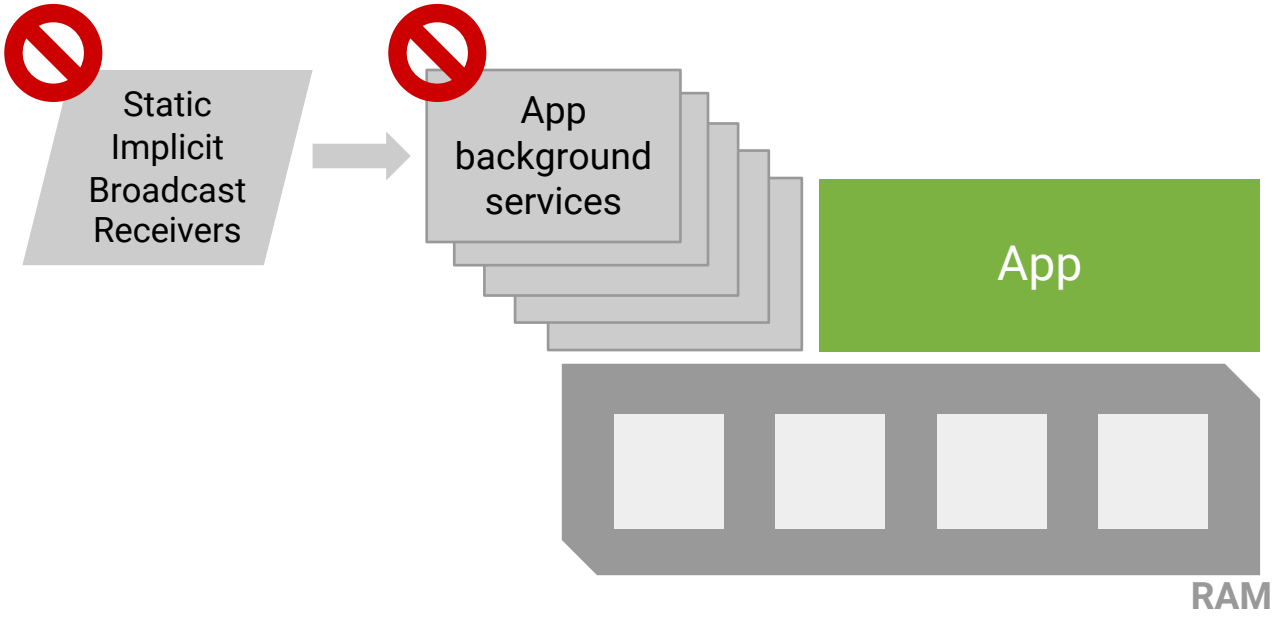
例如，系统常常会播出 connectivity_change 这个 implicit broadcast。



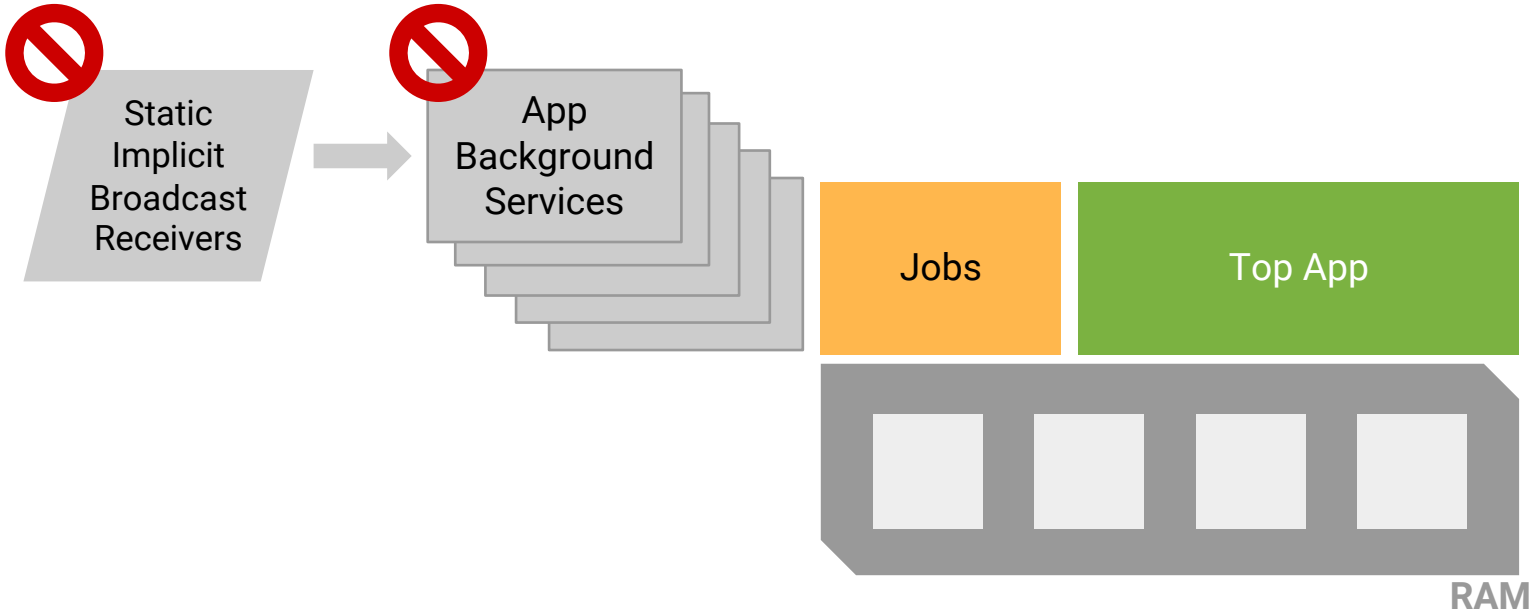
*低内存设备*上的内存抖动



解決方案（!?）


Static
Implicit
Broadcast
Receivers
App
background
services
App
RAM

解决方案


Static
Implicit
Broadcast
Receivers
App
Background
Services
Jobs
Top App
RAM

解决方案: 内存优化

App 里所 **statically declare** 的 implicit broadcast receivers 将不再被唤醒。
Explicit broadcasts 不受影响。

implicit:
`CONNECTIVITY_CHANGE`

explicit:
`MY_PACKAGE_REPLACED`

Apps 不能跑 unbound background services。
Foreground services（用户场景如用来播放音乐，等等） 不受影响。

`startService()`
`bindService()`

路线图


现在 (Android Nougat)
开发者可以测试低内存的状况和优化
消除三个 broadcasts
未来
更多的优化（适用于 target 新版本的 apps）

在Android Nougat 里测试低内存优化

开始测试:

```
$ adb shell cmd appops set <package> RUN_IN_BACKGROUND ignore
```

停止测试:

```
$ adb shell cmd appops set <package> RUN_IN_BACKGROUND allow
```

https://developer.android.com/topic/performance/background-optimization.html

Android Nougat 里到底改了什么?

内存优化

CONNECTIVITY_CHANGE

凡是 target Nougat 的 Apps will 将不会再被这些 static broadcasts 所唤醒。

`android.net.conn.CONNECTIVITY_CHANGE`

在这些情况下，应该使用 `JobScheduler` 的 network triggers。

运行时所 declare 的 receiver （就是 **`Context.registerReceiver()`**）照样会听取到这些广播。

http://developer.android.com/preview/features/background-optimization.html

NEW_PICTURE
NEW_VIDEO

android.hardware.action.NEW_PICTURE
android.hardware.action.NEW_VIDEO

全部 apps (不只是 target Nougat 的) 将不能再发送或听取这些 broadcasts。

JobScheduler 已经增加了听取 **content provider 的改动而启动的功能。**

http://developer.android.com/preview/features/background-optimization.html


APIs

JobScheduler

JobScheduler

延迟 合并

- 包装着 background activity 的 API

- 基于开发者设置的触发类别:
  - 时间窗口
  - 网络类型
  - 设备充电状况
  - 设备的动静态

- 系统利用这些触发的设置，合并相同的 background process，从而优化内存和电池的性能

http://developer.android.com/reference/android/app/job/JobScheduler.html

例子：每 24 小时上载到服务器

```
JobInfo uploadJob =
    new JobInfo.Builder(mSomeInt, mServiceComponent)
    .setRequiredNetworkCapabilities(JobInfo.NETWORK_TYPE_UNMETERED)
    .setPeriodic(24 * DateUtils.HOURS_IN_MILLIS)
    .setRequiresCharging(true)
    .build();

mJobScheduler.schedule(uploadJob);
```

JobScheduler

延迟

合并

- 新增：听取 **Content Provider** 的改动进行触发
- 优先处理 foreground 应用的 jobs
- 同时运行更多 jobs，并根据 RAM 可用性进行扩展。

http://developer.android.com/preview/features/background-optimization.html

使用 JobScheduler，代替了 NEW_PICTURE

```
public static void scheduleJob(Context context) {
  JobScheduler js = (JobScheduler) context.getSystemService(JobScheduler.class);
  JobInfo.Builder builder = new JobInfo.Builder(
          R.id.schedule_photos_job,
          new ComponentName(context, PhotoContentJob.class));
  builder.addTriggerContentUri(
          new
JobInfo.TriggerContentUri(MediaStore.Images.Media.EXTERNAL_CONTENT_URI,
              JobInfo.TriggerContentUri.FLAG_NOTIFY_FOR_DESCENDANTS));
  js.schedule(builder.build());
}
```

我的 app 需要在 API 21 前操作！怎么办？


Firebase JobDispatcher
(旧名称：GCMNetworkManager)

Firebase JobDispatcher

- 新的开源 **SDK**，适用于在Android上调度和执行 background process，也即将支持 iOS！
- 提供大部分 JobScheduler 的功能，帮助向后兼容。

Android L 之前	Android L＋之后	iOS (即将到来)
Google Play Services 拥有中央排程驱动程式	使用 JobScheduler	将使用 Grand Central Dispatch，提供高级词汇来描述你的 jobs

github.com/firebase/firebase-jobdispatcher-android

Firebase JobDispatcher：简单的 job

```
public class MyJobService extends SimpleJobService {
  public int onRunJob(JobParameters spec) {
    return doWork() ? RESULT_SUCCESS : RESULT_FAIL_RETRY;
  }

  private boolean doWork() { /* job 的代码在这 */ }
}
```

Firebase JobDispatcher: 调度 job

```
FirebaseJobDispatcher dispatcher =
    new FirebaseJobDispatcher(new GooglePlayDriver(myContext));

dispatcher.schedule(dispatcher.newJobBuilder()
    .setService(MyJobService.class)
    .setTag("my-job-service")
    .setConstraints(Constraint.ON_UNMETERED_NETWORK)
    .build());
```

诊断工具

电池

Bug Report 的 Batterystats

Statistics since last charge:

System starts: 0, currently on battery: false

Time on battery: 7h 23m 55s 998ms (99.9%) realtime, 3h 48m 25s 999ms (51.4%) uptime

Time on battery screen off: 6h 32m 34s 839ms (88.3%) realtime, 2h 57m 4s 841ms (39.8%) uptime

Total run time: 7h 24m 30s 554ms realtime, 3h 48m 57s 796ms uptime

Start clock time: 2015-01-27-12-39-35

Screen on: 51m 21s 159ms (11.6%) 34x, Interactive: 51m 3s 2ms (11.5%)

Screen brightnesses:

dark 30m 42s 856ms (59.8%)
dim 4m 8s 744ms (8.1%)
medium 7m 50s 754ms (15.3%)
light 4m 54s 354ms (9.6%)
bright 3m 44s 451ms (7.3%)

Total full wakelock time: 4m 7s 889ms

Total partial wakelock time: 2h 36m 8s 832ms

Mobile total received: 5.97MB, sent: 15.17MB (packets received 15913, sent 21930)

Bug Report 的 Battery History

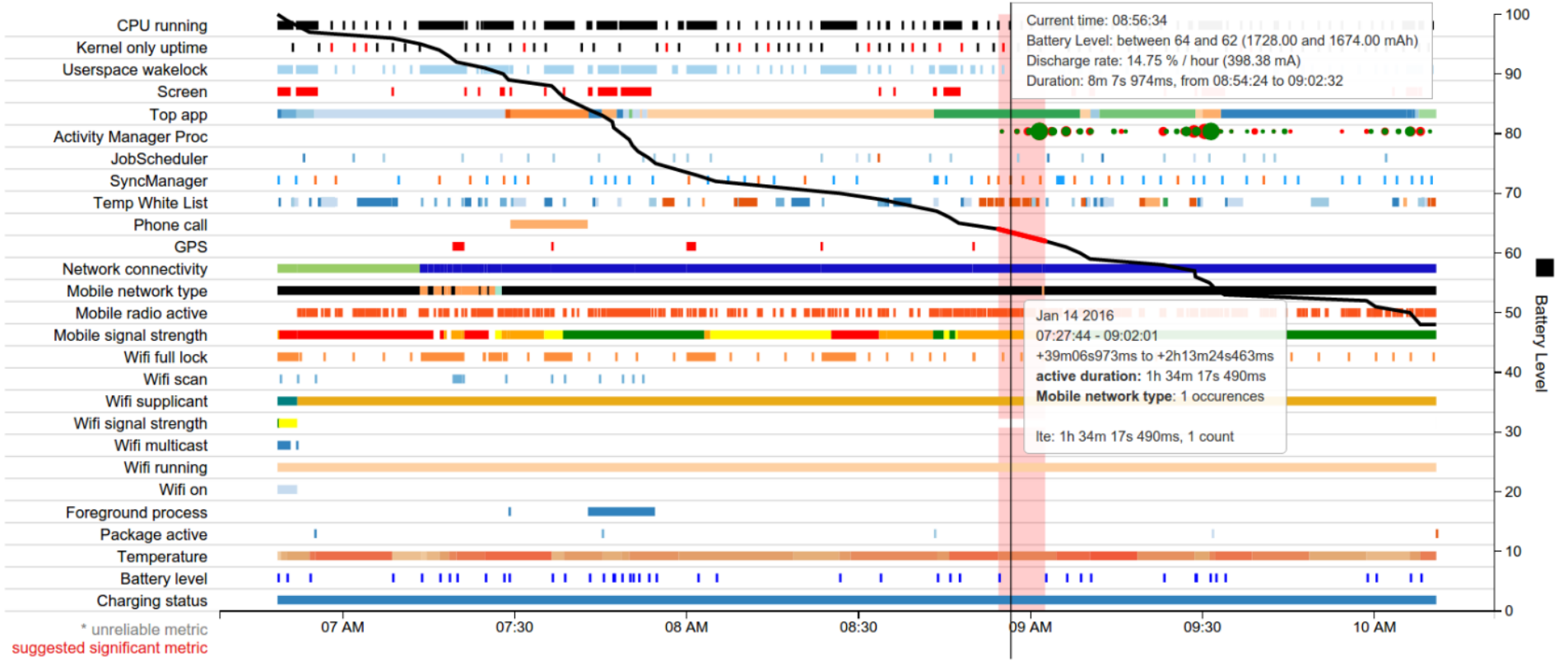
+6m43s466ms (1) 097 d2101a11 +mobile_radio
+6m48s611ms (2) 097 d0101a19 -mobile_radio signal_strength=good
+6m51s655ms (3) 097 d0101a11 signal_strength=moderate -top=u0a52:"com.google.android.talk"
+6m52s150ms (2) 097 d0101a11 +top=u0a22:"com.google.android.googlequicksearchbox"
+6m55s977ms (2) 097 d0101a11 -top=u0a22:"com.google.android.googlequicksearchbox"
+6m55s977ms (2) 097 d2101a11 +mobile_radio +top=u0a56:"com.google.android.apps.maps"
+6m57s250ms (1) 097 f2101a11 +gps

...

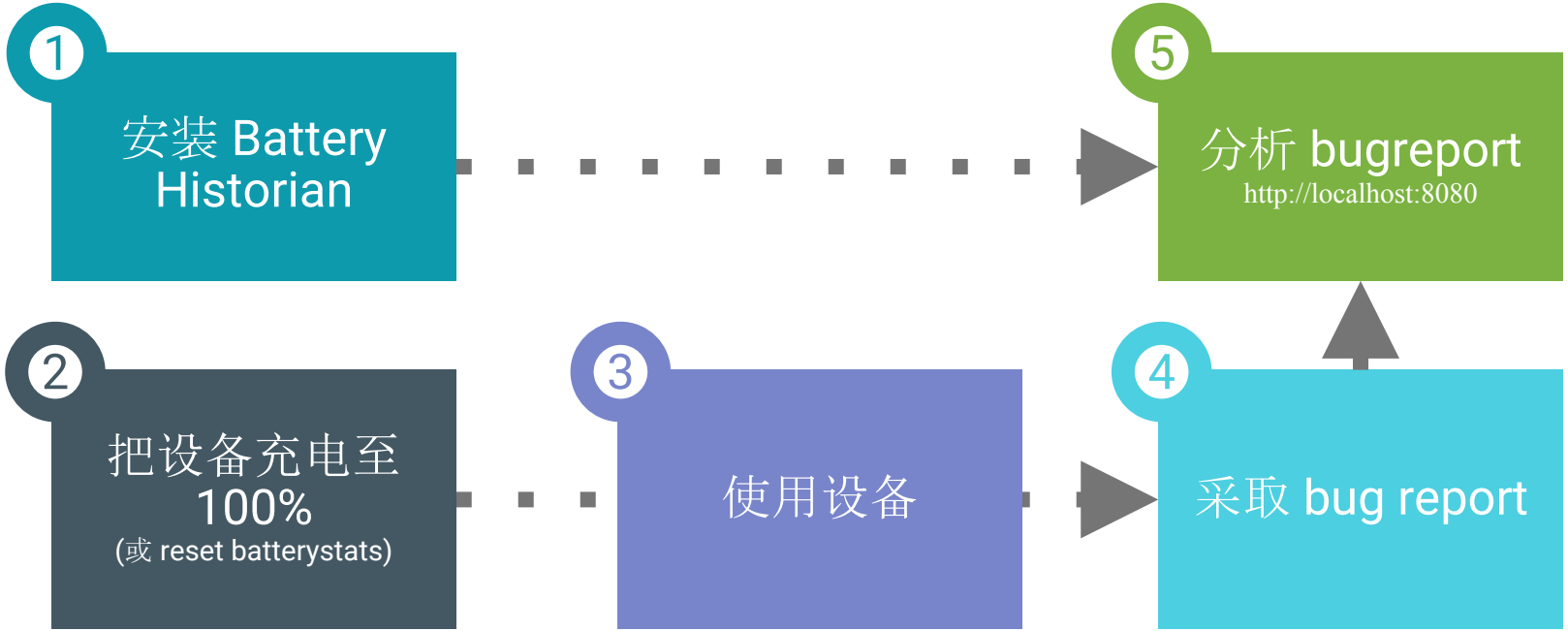
+2h09m05s278ms (2) 084 c2001409 +wake_lock=1000:"SyncLoopWakeLock"
+2h09m05s280ms (1) 084 82001409 -wake_lock
+2h09m05s290ms (2) 084 c2001409 +wake_lock=u0a9:"*net_scheduler*"
+2h09m05s293ms (1) 084 02001409 -running -wake_lock
+2h09m07s775ms (2) 084 c2001409 +running +wake_lock=1001:"RILJ" wake_reason=0:"58:qcom,smsm-r
+2h09m08s302ms (2) 084 82001e09 -wake_lock data_conn

Battery Historian

Historian 时间线

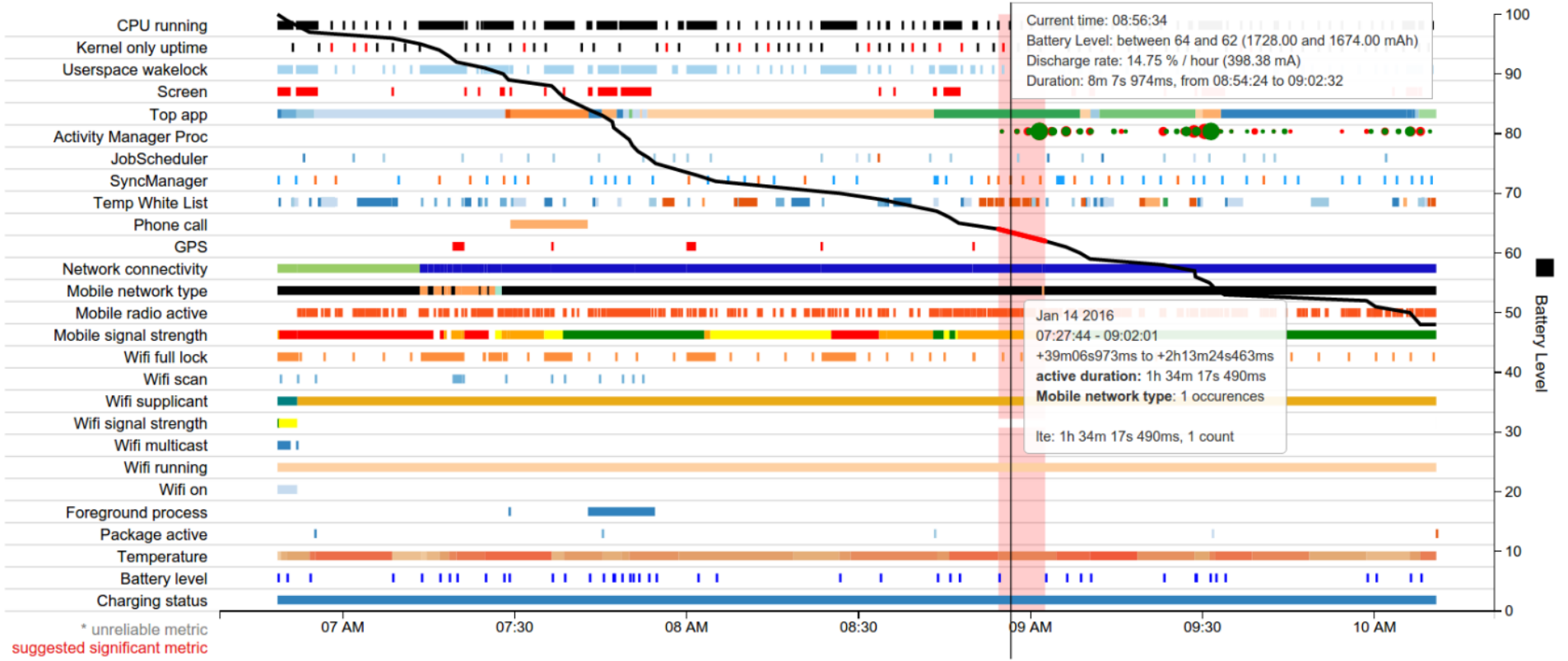


Battery Historian 工作流

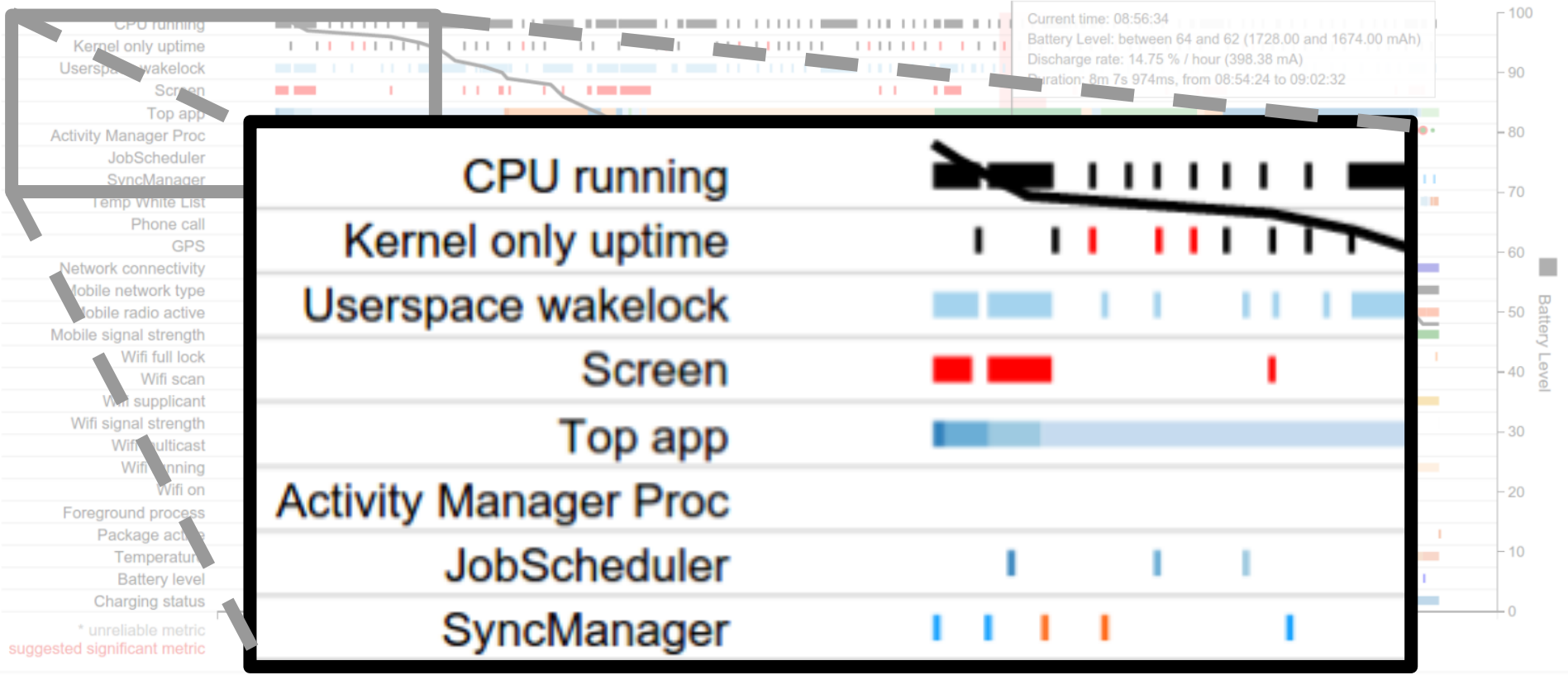


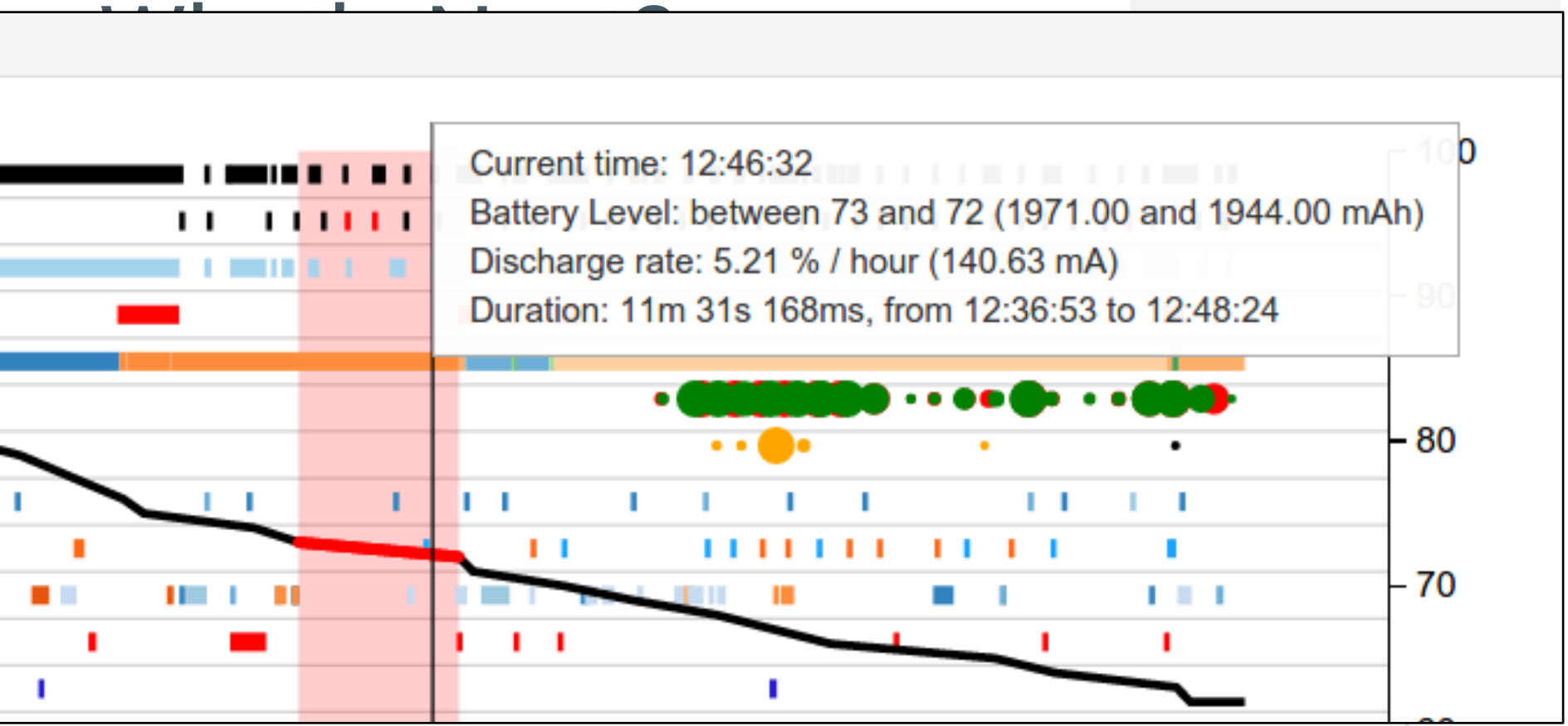
github.com/google/battery-historian

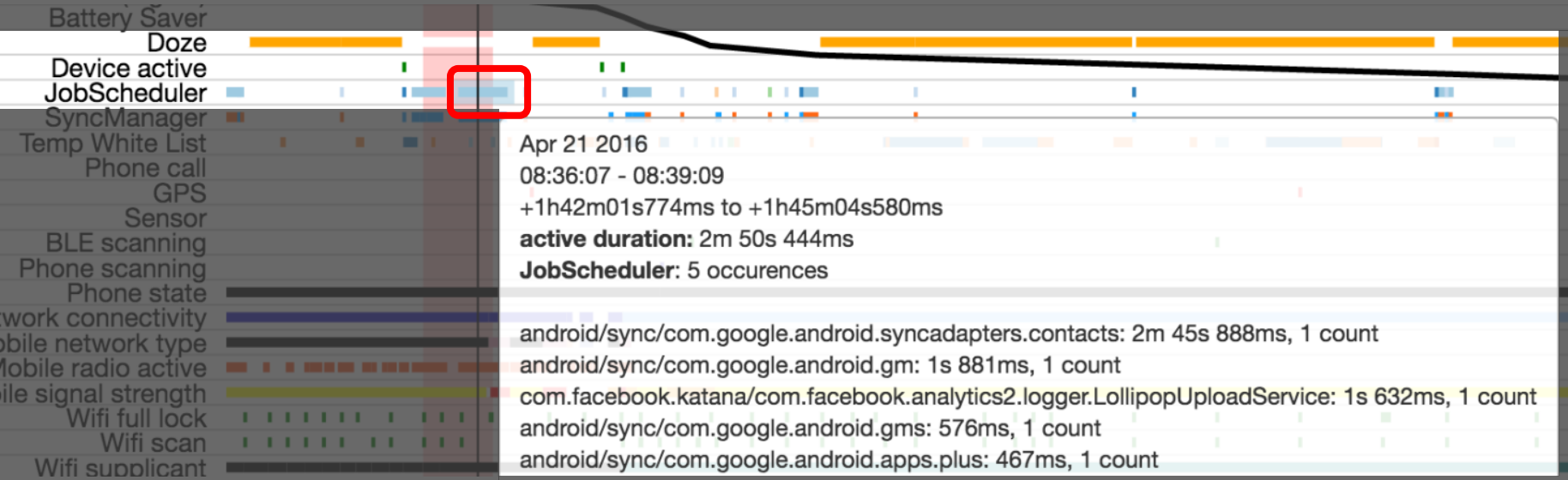
Historian 时间线

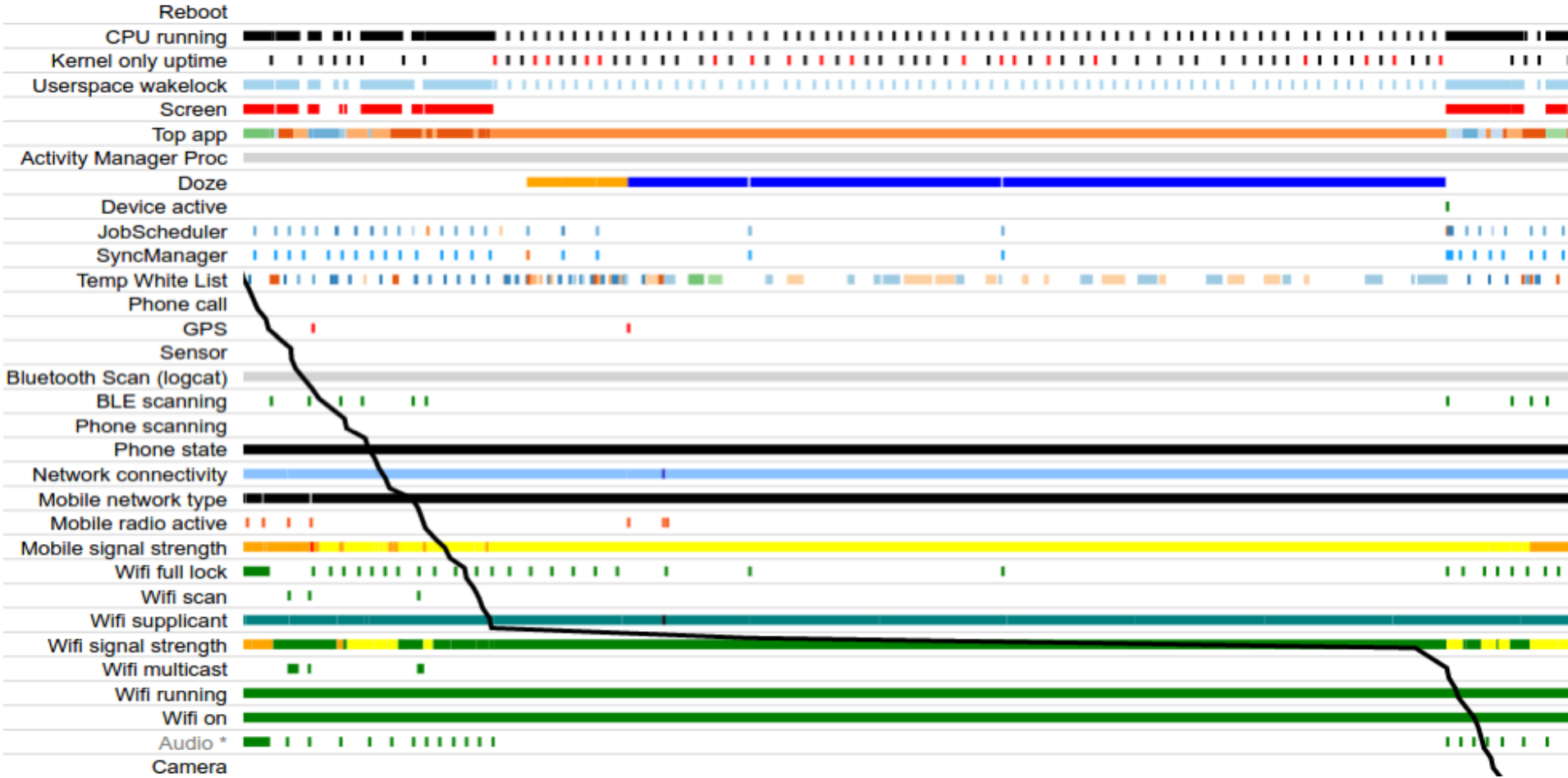


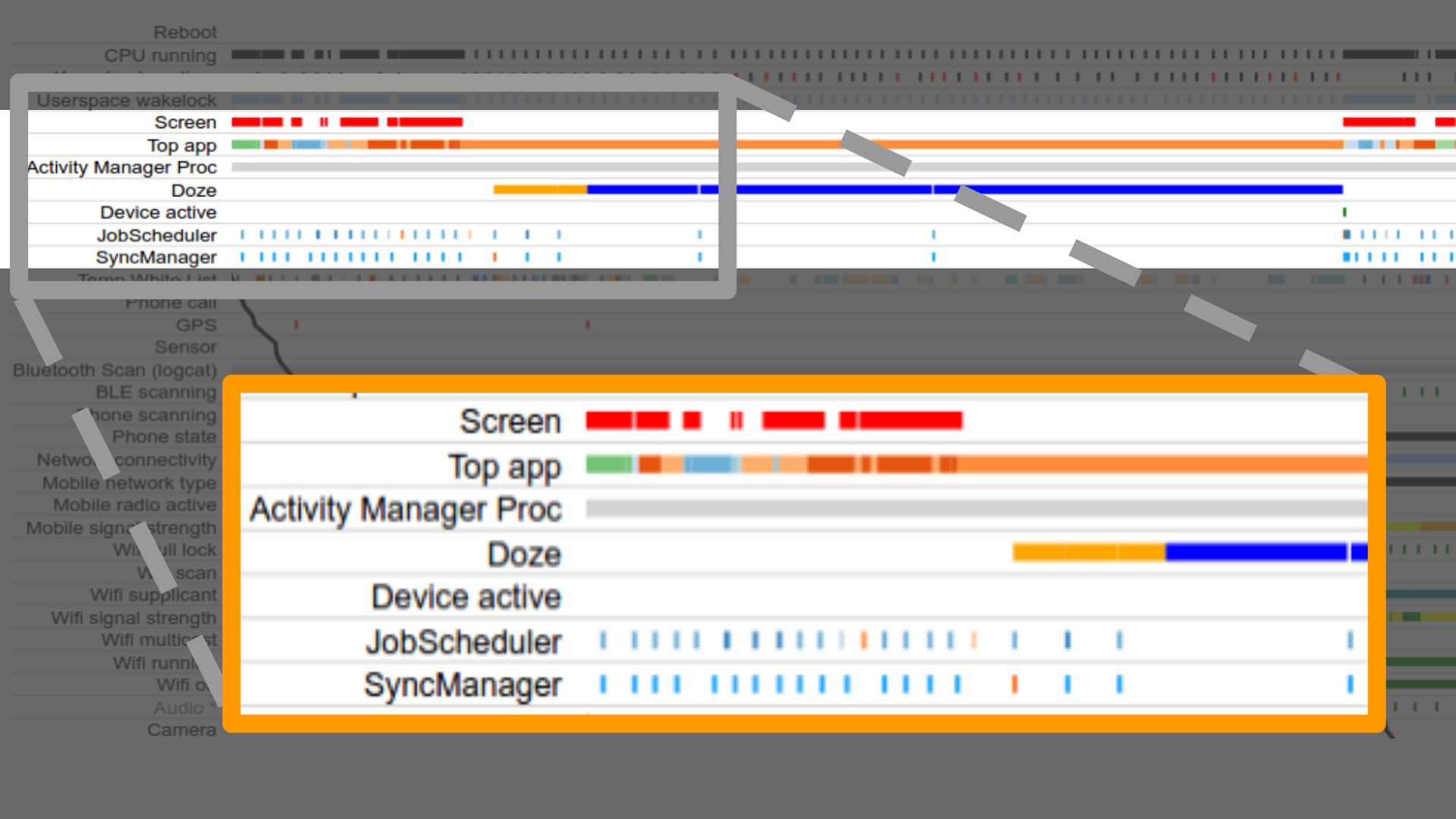
Historian 时间线

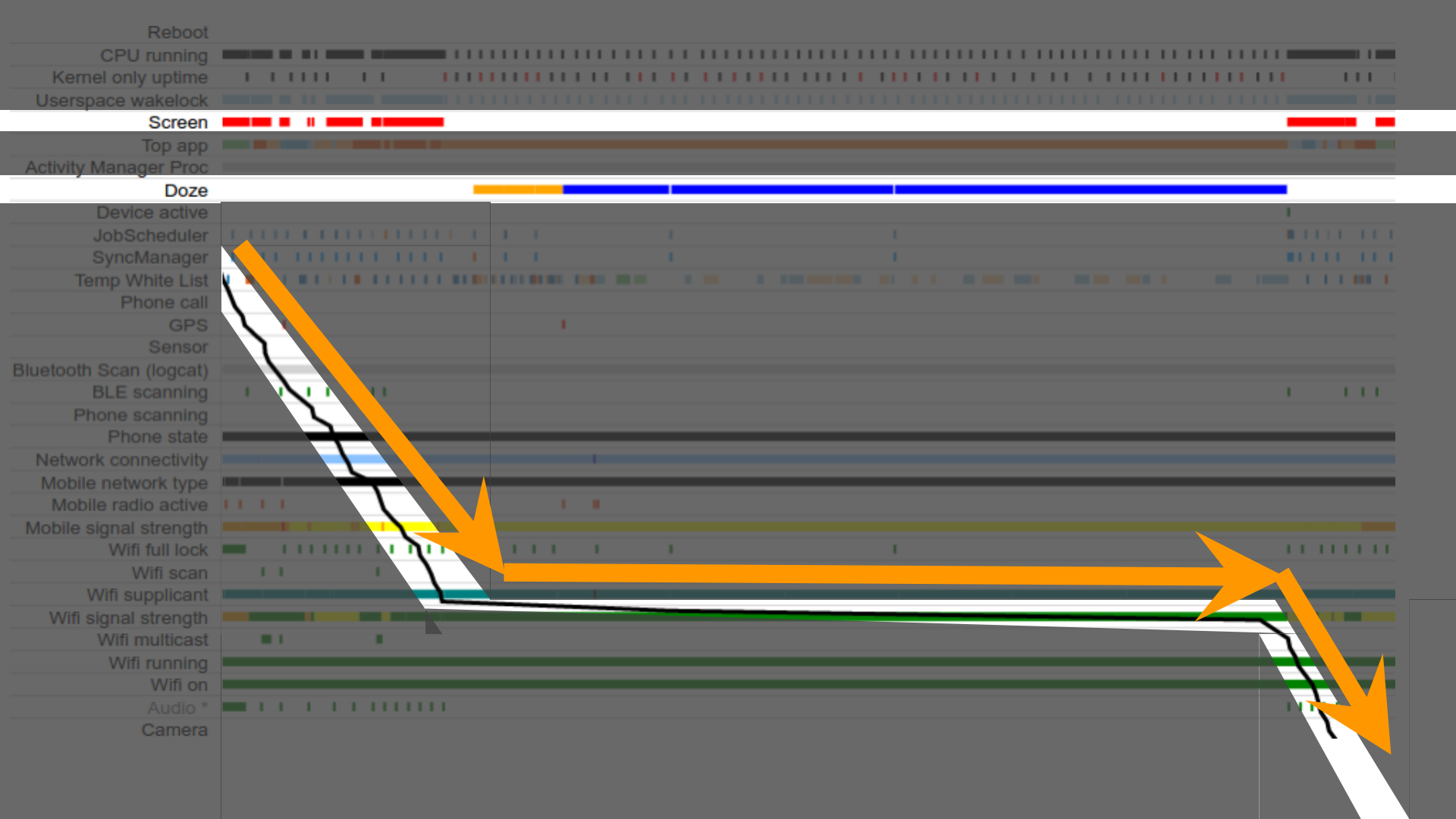



Current time: 12:46:32
Battery Level: between 73 and 72 (1971.00 and 1944.00 mAh)
Discharge rate: 5.21 % / hour (140.63 mA)
Duration: 11m 31s 168ms, from 12:36:53 to 12:48:24
80
70

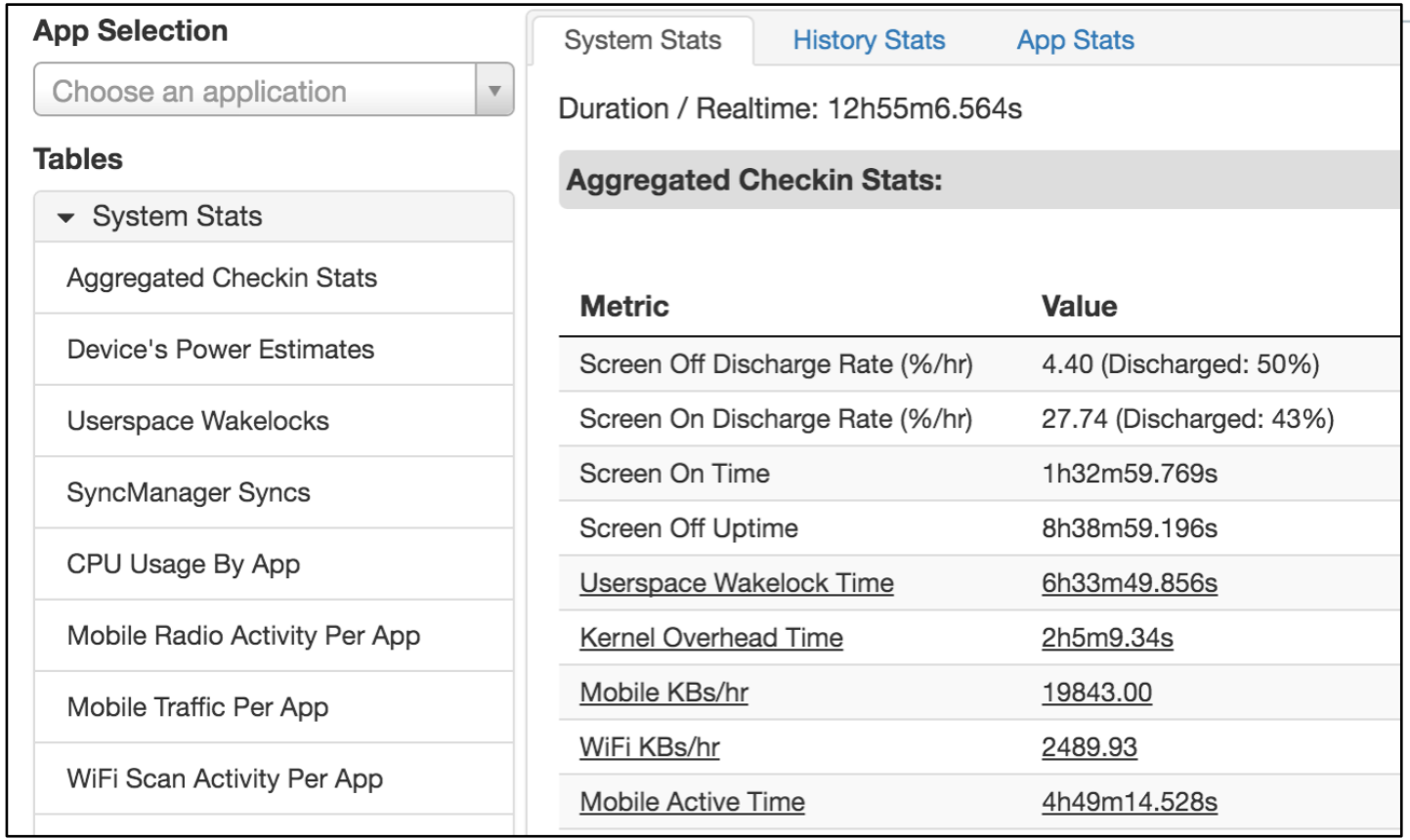


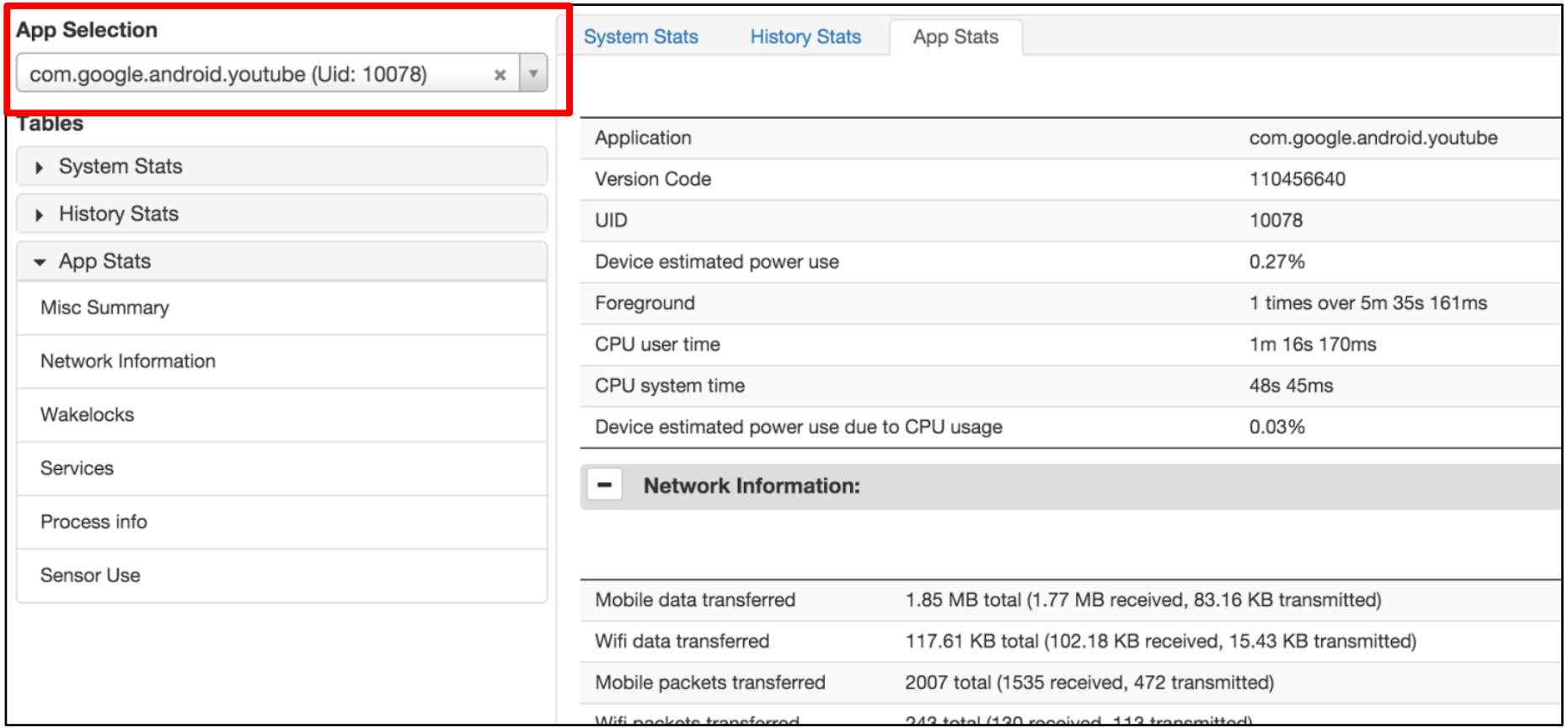

Reboot
CPU running
Kernel only uptime
Userspace wakelock
Screen
Top app
Activity Manager Proc
Doze
Device active
JobScheduler
SyncManager
Temp White List
Phone call
GPS
Sensor
Bluetooth Scan (logcat)
BLE scanning
Phone scanning
Phone state
Network connectivity
Mobile network type
Mobile radio active
Mobile signal strength
Wifi full lock
Wifi scan
Wifi supplicant
Wifi signal strength
Wifi multicast
Wifi running
Wifi on
Audio *
Camera


Reboot
CPU running
Userspace wakelock
Screen
Top app
Activity Manager Proc
Doze
Device active
JobScheduler
SyncManager
Phone call
GPS
Sensor
Bluetooth Scan (logcat)
BLE scanning
Phone state
Mobile network type
Mobile radio active
Wifi supplicant
Wifi signal strength
Camera
Screen
Top app
Activity Manager Proc
Doze
Device active
JobScheduler
SyncManager

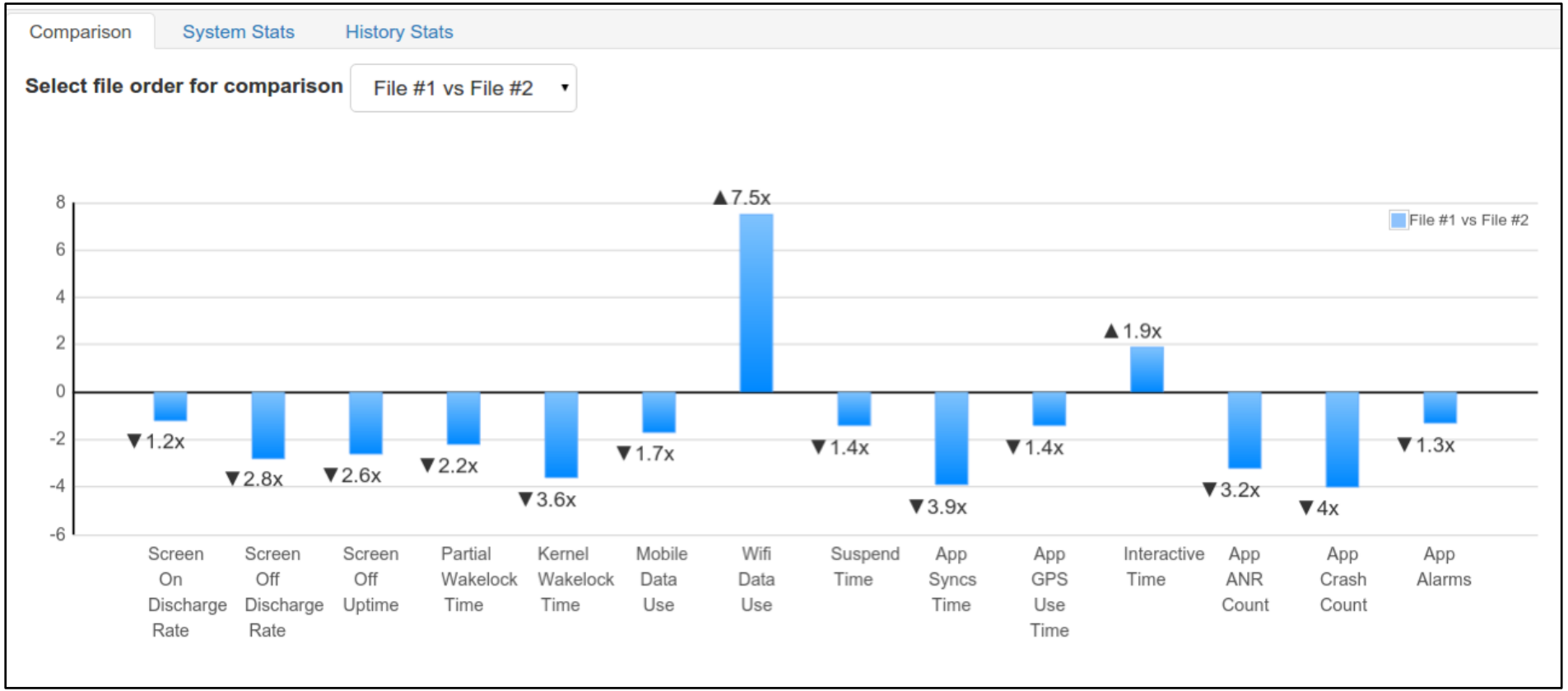

Reboot
CPU running
Kernel only uptime
Userspace wakelock
Screen
Top app
Activity Manager Proc
Doze
Device active
JobScheduler
SyncManager
Temp White List
Phone call
GPS
Sensor
Bluetooth Scan (logcat)
BLE scanning
Phone scanning
Phone state
Network connectivity
Mobile network type
Mobile radio active
Mobile signal strength
Wifi full lock
Wifi scan
Wifi supplicant
Wifi signal strength
Wifi multicast
Wifi running
Wifi on
Audio *
Camera

系统统计

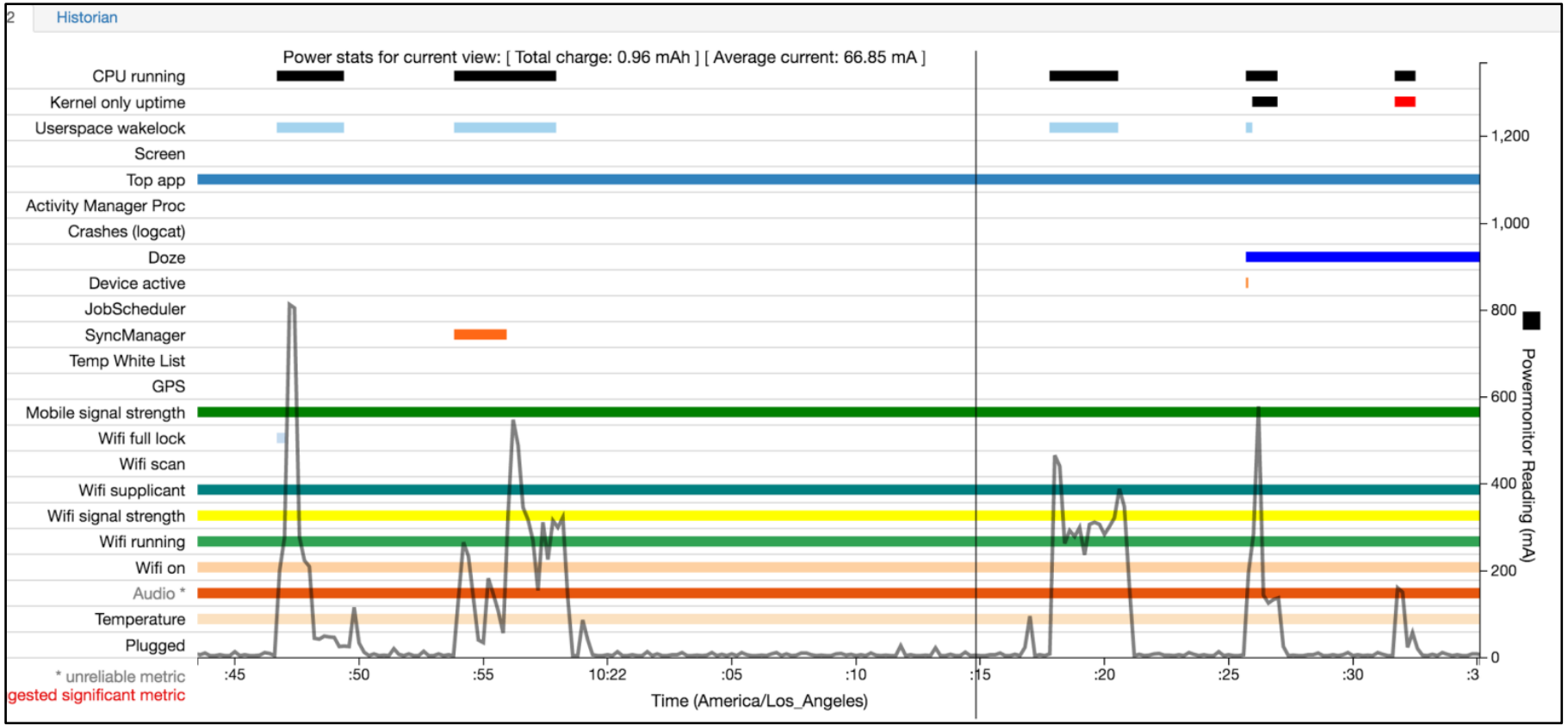

App Selection
Choose an application
Tables
System Stats
Aggregated Checkin Stats
Device's Power Estimates
Userspace Wakelocks
SyncManager Syncs
CPU Usage By App
Mobile Radio Activity Per App
Mobile Traffic Per App
WiFi Scan Activity Per App
System Stats
History Stats
App Stats
Duration / Realtime: 12h55m6.564s
Aggregated Checkin Stats:
Metric
Value
Screen Off Discharge Rate (%/hr)
4.40 (Discharged: 50%)
Screen On Discharge Rate (%/hr)
27.74 (Discharged: 43%)
Screen On Time
1h32m59.769s
Screen Off Uptime
8h38m59.196s
Userspace Wakelock Time
6h33m49.856s
Kernel Overhead Time
2h5m9.34s
Mobile KBs/hr
19843.00
WiFi KBs/hr
2489.93
Mobile Active Time
4h49m14.528s


App Selection
com.google.android.youtube (Uid: 10078)
Tables
System Stats
History Stats
App Stats
Misc Summary
Network Information
Wakelocks
Services
Process info
Sensor Use
System Stats
History Stats
App Stats
Application
com.google.android.youtube
Version Code
110456640
UID
10078
Device estimated power use
0.27%
Foreground
1 times over 5m 35s 161ms
CPU user time
1m 16s 170ms
CPU system time
48s 45ms
Device estimated power use due to CPU usage
0.03%
Network Information:
Mobile data transferred
1.85 MB total (1.77 MB received, 83.16 KB transmitted)
Wifi data transferred
117.61 KB total (102.18 KB received, 15.43 KB transmitted)
Mobile packets transferred
2007 total (1535 received, 472 transmitted)

A/B 比较


Comparison
System Stats
History Stats
Select file order for comparison
File #1 vs File #2
File #1 vs File #2
8
6
4
2
0
-2
-4
-6
▼1.2x
▼2.8x
▼2.6x
▼2.2x
▼3.6x
▼1.7x
▲7.5x
▼1.4x
▼3.9x
▼1.4x
▲1.9x
▼3.2x
▼4x
▼1.3x
Screen On Discharge Rate
Screen Off Discharge Rate
Screen Off Uptime
Partial Wakelock Time
Kernel Wakelock Time
Mobile Data Use
Wifi Data Use
Suspend Time
App Syncs Time
App GPS Use Time
Interactive Time
App ANR Count
App Crash Count
App Alarms

Powermonitor reading overlay



下一步

减少 | 延迟 | 合并

去除对以下的依赖：

- statically declared implicit broadcast receivers
- unbound background services

使用 `JobScheduler` 和 `Firebase JobDispatcher`:

- developer.android.com/reference/android/app/job/JobScheduler.html
- github.com/firebase/firebase-jobdispatcher-android

使用 Battery Historian:

- github.com/google/battery-historian

谢谢！