# Towards Verifying Android Apps for the Absence of No-Sleep Energy Bugs

Martín Carreiro - Pablo Rago - Juan Manuel Tastzian

Facultad de Ciencias Exactas y Naturales

21 de Mayo de 2014

# Ventajas de los smartphones actuales

Muchas características al alcance de la mano:

# Ventajas de los smartphones actuales

Muchas características al alcance de la mano:

- Procesadores rápidos

# Ventajas de los smartphones actuales

Muchas características al alcance de la mano:

- Procesadores rápidos
- Pantallas grandes

# Ventajas de los smartphones actuales

Muchas características al alcance de la mano:

- Procesadores rápidos
- Pantallas grandes
- Cámara, GPS, 3G, Wifi...

# ¿Y antes?

Pero...

# ¿Y antes?

Pero...

...¿se acuerdan del pasado?

# ¡LEYENDA!

# ¿Cuánto les duraba la batería del Nokia 1100 y similares?

# ¿Cuánto les duraba la batería del Nokia 1100 y similares?

- ¿15 hs?

# ¿Cuánto les duraba la batería del Nokia 1100 y similares?

- ¿15 hs?
- ¿2 días?

# ¿Cuánto les duraba la batería del Nokia 1100 y similares?

- ¿15 hs?
- ¿2 días?
- Como 4 o 5, tranqui!

# Eso ya no pasa con los Smartphones

# Eso ya no pasa con los Smartphones

Lamentablemente, todas las ventajas mencionadas antes necesitan mucha energía para funcionar.

# Eso ya no pasa con los Smartphones

Lamentablemente, todas las ventajas mencionadas antes necesitan mucha energía para funcionar.

Y por eso, la batería de nuestros celulares de hoy dura entre 12 y 16 horas, promedio, siendo unas 18 o 20 horas una excelente duración de batería (ni hablar de más tiempo).

# Manejo de energía en Android

¿Cómo maneja la energía Android?

# Manejo de energía en Android

¿Cómo maneja la energía Android?

- De manera **agresiva**.

# Manejo de energía en Android

¿Cómo maneja la energía Android?

- De manera **agresiva**.

¿Le pega para que se porte bien?

# Manejo de energía en Android

¿Cómo maneja la energía Android?

- De manera **agresiva**.

¿Le pega para que se porte bien?

- No **tan** agresiva, pero le corta el chorro a todo en el momento **inmediato** en el que se deja de usar.

# Manejo de energía en Android

Pero, ¿y si, como desarrollador, quiero tener el procesador corriendo para recibir algún update o notificación?

# Manejo de energía en Android

Pero, ¿y si, como desarrollador, quiero tener el procesador corriendo para recibir algún update o notificación?

- Ahí es donde entra en juego la **Wakelock API**.

# Manejo de energía en Android

Lamentablemente, la complejidad del sistema operativo y los errores que pueden cometer los desarrolladores hacen que el uso inapropiado de Wakelocks se manifiesten como *no-sleep-bugs*.

# Manejo de energía en Android

Lamentablemente, la complejidad del sistema operativo y los errores que pueden cometer los desarrolladores hacen que el uso inapropiado de Wakelocks se manifiesten como *no-sleep-bugs*.

Los autores del paper decidieron intentar mitigar el problema implementando una herramienta que verifica la ausencia de estos bugs con respecto a una serie de políticas específicas sobre los Wakelocks, utilizando un framework de flujo de datos para analizar las aplicaciones.

# Manejo de energía en Android

Lamentablemente, la complejidad del sistema operativo y los errores que pueden cometer los desarrolladores hacen que el uso inapropiado de Wakelocks se manifiesten como *no-sleep-bugs*.

Los autores del paper decidieron intentar mitigar el problema implementando una herramienta que verifica la ausencia de estos bugs con respecto a una serie de políticas específicas sobre los Wakelocks, utilizando un framework de flujo de datos para analizar las aplicaciones.

Pero...

# Manejo de energía en Android

Lamentablemente, la complejidad del sistema operativo y los errores que pueden cometer los desarrolladores hacen que el uso inapropiado de Wakelocks se manifiesten como *no-sleep-bugs*.

Los autores del paper decidieron intentar mitigar el problema implementando una herramienta que verifica la ausencia de estos bugs con respecto a una serie de políticas específicas sobre los Wakelocks, utilizando un framework de flujo de datos para analizar las aplicaciones.

Pero...

¿qué es la Wakelock API?

# Wakelock API

# Wakelock API

- Permite a los desarrolladores dar directivas específicas sobre los recursos al sistema operativo, ya que Android pone todo en sleep mode ni bien se ponen en estado idle (reposo).

# Wakelock API

- Permite a los desarrolladores dar directivas específicas sobre los recursos al sistema operativo, ya que Android pone todo en sleep mode ni bien se ponen en estado idle (reposo).
- Es una forma de decir "esto no me lo apagues" (GPS, pantalla, CPU, etc.).

# Wakelock API

- Permite a los desarrolladores dar directivas específicas sobre los recursos al sistema operativo, ya que Android pone todo en sleep mode ni bien se ponen en estado idle (reposo).
- Es una forma de decir "esto no me lo apagues" (GPS, pantalla, CPU, etc.).
- Entonces si necesito que algo en particular esté encendido en un momento crítico, pido un Wakelock sobre el mismo para que no se apague, lo uso, y cuando termino, lo libero.

# Wakelock API

- Permite a los desarrolladores dar directivas específicas sobre los recursos al sistema operativo, ya que Android pone todo en sleep mode ni bien se ponen en estado idle (reposo).
- Es una forma de decir "esto no me lo apagues" (GPS, pantalla, CPU, etc.).
- Entonces si necesito que algo en particular esté encendido en un momento crítico, pido un Wakelock sobre el mismo para que no se apague, lo uso, y cuando termino, lo libero.
- Genial! Pero ¿qué pasa si me olvido de liberarlo?

# Wakelock API

- Permite a los desarrolladores dar directivas específicas sobre los recursos al sistema operativo, ya que Android pone todo en sleep mode ni bien se ponen en estado idle (reposo).
- Es una forma de decir "esto no me lo apagues" (GPS, pantalla, CPU, etc.).
- Entonces si necesito que algo en particular esté encendido en un momento crítico, pido un Wakelock sobre el mismo para que no se apague, lo uso, y cuando termino, lo libero.
- Genial! Pero ¿qué pasa si me olvido de liberarlo?
- Buena pregunta... ¿cuánto se acuerdan de Orga 2?

# Repaso de Orga 2

# Repaso de Orga 2

- En Orga 2 me enseñaron que debo hacer un free por cada malloc...

# Repaso de Orga 2

- En Orga 2 me enseñaron que debo hacer un free por cada malloc...
  - ...sino me pegaban.

# Repaso de Orga 2

- En Orga 2 me enseñaron que debo hacer un free por cada malloc...
  - ...sino me pegaban.
- Hablando en serio: si no liberan el Wakelock, pasa lo que esperan.

# Repaso de Orga 2

- En Orga 2 me enseñaron que debo hacer un free por cada malloc...
  - ...sino me pegaban.
- Hablando en serio: si no liberan el Wakelock, pasa lo que esperan.
- El recurso no se libera, queda activo, y se **gasta batería** innecesariamente.

# Repaso de Orga 2

- En Orga 2 me enseñaron que debo hacer un free por cada malloc...
  - ...sino me pegaban.
- Hablando en serio: si no liberan el Wakelock, pasa lo que esperan.
- El recurso no se libera, queda activo, y se **gasta batería** innecesariamente.
- Esto es a lo que en el paper se lo llama *no-sleep bug*.

# Repaso de Orga 2

- En Orga 2 me enseñaron que debo hacer un free por cada malloc...
  - ...sino me pegaban.
- Hablando en serio: si no liberan el Wakelock, pasa lo que esperan.
- El recurso no se libera, queda activo, y se **gasta batería** innecesariamente.
- Esto es a lo que en el paper se lo llama *no-sleep bug*.
- Vendría a ser la versión de "power management" de los "memory leaks".

# Planteo del paper

# Planteo del paper

Los autores del paper desarrollaron una herramienta que permite verificar la ausencia de estos bugs con respecto a una serie de **políticas específicas sobre los Wakelocks**, utilizando un framework de flujo de datos para analizar las aplicaciones.

# Planteo del paper

Los autores del paper desarrollaron una herramienta que permite verificar la ausencia de estos bugs con respecto a una serie de **políticas específicas sobre los Wakelocks**, utilizando un framework de flujo de datos para analizar las aplicaciones.

Lo peor de todo esto, es que estos bugs son muy difíciles de detectar, ya que no hace que la app funcione mal o crashee, sino que te reduce la duración de batería del equipo, pasando casi desapercibida como causante de dicho problema.

# Planteo del paper

Los autores del paper desarrollaron una herramienta que permite verificar la ausencia de estos bugs con respecto a una serie de **políticas específicas sobre los Wakelocks**, utilizando un framework de flujo de datos para analizar las aplicaciones.

Lo peor de todo esto, es que estos bugs son muy difíciles de detectar, ya que no hace que la app funcione mal o crashee, sino que te reduce la duración de batería del equipo, pasando casi desapercibida como causante de dicho problema.

Pero para entender todo esto, primero hay que entender un poco como funciona el sistema operativo Android por detrás.

# Un poco de contexto

## Manejo de energía

# Un poco de contexto

## Manejo de energía

- Creación, adquisición y liberación de objetos **Wakelock**.

# Un poco de contexto

## Manejo de energía

- Creación, adquisición y liberación de objetos **Wakelock**.
- Asociados a un recurso particular (CPU, pantalla, GPS, etc.).

# Un poco de contexto

## Manejo de energía

- Creación, adquisición y liberación de objetos **Wakelock**.
- Asociados a un recurso particular (CPU, pantalla, GPS, etc.).
- En manos del desarrolador.

# Un poco de contexto

## Manejo de energía

- Creación, adquisición y liberación de objetos **Wakelock**.
- Asociados a un recurso particular (CPU, pantalla, GPS, etc.).
- En manos del desarrolador.

## Componentes de aplicación

# Un poco de contexto

## Manejo de energía

- Creación, adquisición y liberación de objetos **Wakelock**.
- Asociados a un recurso particular (CPU, pantalla, GPS, etc.).
- En manos del desarrolador.

## Componentes de aplicación

- Una aplicación de Android se construye con diversos componentes. Hay 4 tipos principales.

# Un poco de contexto

## Manejo de energía

- Creación, adquisición y liberación de objetos **Wakelock**.
- Asociados a un recurso particular (CPU, pantalla, GPS, etc.).
- En manos del desarrolador.

## Componentes de aplicación

- Una aplicación de Android se construye con diversos componentes. Hay 4 tipos principales.
  - ★ Activities

# Un poco de contexto

## Manejo de energía

- Creación, adquisición y liberación de objetos **Wakelock**.
- Asociados a un recurso particular (CPU, pantalla, GPS, etc.).
- En manos del desarrolador.

## Componentes de aplicación

- Una aplicación de Android se construye con diversos componentes. Hay 4 tipos principales.
  - ★ Activities
  - ★ Services

# Un poco de contexto

## Manejo de energía

- Creación, adquisición y liberación de objetos **Wakelock**.
- Asociados a un recurso particular (CPU, pantalla, GPS, etc.).
- En manos del desarrolador.

## Componentes de aplicación

- Una aplicación de Android se construye con diversos componentes. Hay 4 tipos principales.
  - ★ Activities
  - ★ Services
  - ★ BroadcastReceivers

# Un poco de contexto

## Manejo de energía

- Creación, adquisición y liberación de objetos **Wakelock**.
- Asociados a un recurso particular (CPU, pantalla, GPS, etc.).
- En manos del desarrolador.

## Componentes de aplicación

- Una aplicación de Android se construye con diversos componentes. Hay 4 tipos principales.
  - ★ Activities
  - ★ Services
  - ★ BroadcastReceivers
  - ★ ContentProviders

# Un poco de contexto

## Manejo de energía

- Creación, adquisición y liberación de objetos **Wakelock**.
- Asociados a un recurso particular (CPU, pantalla, GPS, etc.).
- En manos del desarrolador.

## Componentes de aplicación

- Una aplicación de Android se construye con diversos componentes. Hay 4 tipos principales.
  - ★ Activities
  - ★ Services
  - ★ BroadcastReceivers
  - ★ ContentProviders
- Están ligados entre sí mediante un *lifecycle* o ciclo de vida (visto en clase).

# Un poco de contexto

## Manejo de energía

- Creación, adquisición y liberación de objetos **Wakelock**.
- Asociados a un recurso particular (CPU, pantalla, GPS, etc.).
- En manos del desarrolador.

## Componentes de aplicación

- Una aplicación de Android se construye con diversos componentes. Hay 4 tipos principales.
  - ★ Activities
  - ★ Services
  - ★ BroadcastReceivers
  - ★ ContentProviders
- Están ligados entre sí mediante un *lifecycle* o ciclo de vida (visto en clase).
- El desarrollador especifica las acciones a tomar en cada paso del ciclo, implementando un conjunto de **callbacks**.

# Un poco de contexto

## Manejo de energía

- Creación, adquisición y liberación de objetos **Wakelock**.
- Asociados a un recurso particular (CPU, pantalla, GPS, etc.).
- En manos del desarrolador.

## Componentes de aplicación

- Una aplicación de Android se construye con diversos componentes. Hay 4 tipos principales.
  - ★ Activities
  - ★ Services
  - ★ BroadcastReceivers
  - ★ ContentProviders
- Están ligados entre sí mediante un *lifecycle* o ciclo de vida (visto en clase).
- El desarrollador especifica las acciones a tomar en cada paso del ciclo, implementando un conjunto de **callbacks**.
- Veamos que hace cada componente.

# Activities

- Proveen la UI de la app.

# Activities

- Proveen la UI de la app.
- Forman un stack, formado de la siguiente manera:

# Activities

- Proveen la UI de la app.
- Forman un stack, formado de la siguiente manera:
  - **Running:** La aplicación en *foreground* (visible, activa).

# Activities

- Proveen la UI de la app.
- Forman un stack, formado de la siguiente manera:
  - **Running:** La aplicación en *foreground* (visible, activa).
  - **Paused:** La aplicación no está en *foreground*, pero está visible (inactiva).

# Activities

- Proveen la UI de la app.
- Forman un stack, formado de la siguiente manera:
  - **Running:** La aplicación en *foreground* (visible, activa).
  - **Paused:** La aplicación no está en *foreground*, pero está visible (inactiva).
  - **Stopped:** La aplicación está en *background* (no visible e inactiva).

# Activities

- Proveen la UI de la app.
- Forman un stack, formado de la siguiente manera:
  - **Running:** La aplicación en *foreground* (visible, activa).
  - **Paused:** La aplicación no está en *foreground*, pero está visible (inactiva).
  - **Stopped:** La aplicación está en *background* (no visible e inactiva).

**Callbacks**

# Activities

- Proveen la UI de la app.
- Forman un stack, formado de la siguiente manera:
  - **Running:** La aplicación en *foreground* (visible, activa).
  - **Paused:** La aplicación no está en *foreground*, pero está visible (inactiva).
  - **Stopped:** La aplicación está en *background* (no visible e inactiva).

**Callbacks**

- **onPause** y **onResume**: entra y sale del estado *running*.

# Activities

- Proveen la UI de la app.
- Forman un stack, formado de la siguiente manera:
  - **Running:** La aplicación en *foreground* (visible, activa).
  - **Paused:** La aplicación no está en *foreground*, pero está visible (inactiva).
  - **Stopped:** La aplicación está en *background* (no visible e inactiva).

**Callbacks**

- **onPause** y **onResume**: entra y sale del estado *running*.
- **onStart** y **onStop**: entra y sale del estado *paused*.

# Activities

- Proveen la UI de la app.
- Forman un stack, formado de la siguiente manera:
  - **Running:** La aplicación en *foreground* (visible, activa).
  - **Paused:** La aplicación no está en *foreground*, pero está visible (inactiva).
  - **Stopped:** La aplicación está en *background* (no visible e inactiva).

**Callbacks**

- **onPause** y **onResume**: entra y sale del estado *running*.
- **onStart** y **onStop**: entra y sale del estado *paused*.
- **onCreate** y **onDestroy**: arranca y termina el *lifecycle*.

# Activities

- Proveen la UI de la app.
- Forman un stack, formado de la siguiente manera:
  - **Running:** La aplicación en *foreground* (visible, activa).
  - **Paused:** La aplicación no está en *foreground*, pero está visible (inactiva).
  - **Stopped:** La aplicación está en *background* (no visible e inactiva).

**Callbacks**

- **onPause** y **onResume**: entra y sale del estado *running*.
- **onStart** y **onStop**: entra y sale del estado *paused*.
- **onCreate** y **onDestroy**: arranca y termina el *lifecycle*.
- **onRestart**: reinicia una actividad previamente detenida.

# Services

- Realizan operaciones de duración prolongada sin interacción del usuario.

# Services

- Realizan operaciones de duración prolongada sin interacción del usuario.
- Se inician con **startService** y permiten ligarse a ellos con **bindService**.

# Services

- Realizan operaciones de duración prolongada sin interacción del usuario.
- Se inician con **startService** y permiten ligarse a ellos con **bindService**.
- Se configuran durante el **onCreate**.

# Services

- Realizan operaciones de duración prolongada sin interacción del usuario.
- Se inician con **startService** y permiten ligarse a ellos con **bindService**.
- Se configuran durante el **onCreate**.
- Se ejecuta **onStartCommand** cuando un servicio es iniciado por otro componente

# Services

- Realizan operaciones de duración prolongada sin interacción del usuario.
- Se inician con **startService** y permiten ligarse a ellos con **bindService**.
- Se configuran durante el **onCreate**.
- Se ejecuta **onStartCommand** cuando un servicio es iniciado por otro componente
- Se ejecuta **onBind** cuando el primer cliente del servicio se liga

# Services

- Realizan operaciones de duración prolongada sin interacción del usuario.
- Se inician con **startService** y permiten ligarse a ellos con **bindService**.
- Se configuran durante el **onCreate**.
- Se ejecuta **onStartCommand** cuando un servicio es iniciado por otro componente
- Se ejecuta **onBind** cuando el primer cliente del servicio se liga
- Se ejecuta **onUnbind** cuando el último cliente del servicio se desliga.

# Services

- Realizan operaciones de duración prolongada sin interacción del usuario.
- Se inician con **startService** y permiten ligarse a ellos con **bindService**.
- Se configuran durante el **onCreate**.
- Se ejecuta **onStartCommand** cuando un servicio es iniciado por otro componente
- Se ejecuta **onBind** cuando el primer cliente del servicio se liga
- Se ejecuta **onUnbind** cuando el último cliente del servicio se desliga.
- Se utiliza **IntentService** para manejar pedidos asincrónicos bajo demanda.

# Services

- Realizan operaciones de duración prolongada sin interacción del usuario.
- Se inician con **startService** y permiten ligarse a ellos con **bindService**.
- Se configuran durante el **onCreate**.
- Se ejecuta **onStartCommand** cuando un servicio es iniciado por otro componente
- Se ejecuta **onBind** cuando el primer cliente del servicio se liga
- Se ejecuta **onUnbind** cuando el último cliente del servicio se desliga.
- Se utiliza **IntentService** para manejar pedidos asincrónicos bajo demanda.
- Al finalizar los callbacks **onStartCommand**, **onUnbind** y **onHandleIntent**, la tarea asociada debería haber terminado, por lo que no debería mantenerse ningún Wakelock.

# BroadcastReceivers

- Responden a anuncios que tienen como alcance a todo el sistema.

# BroadcastReceivers

- Responden a anuncios que tienen como alcance a todo el sistema.
  - Producidos por el mismo sistema (por ejemplo, batería baja)

# BroadcastReceivers

- Responden a anuncios que tienen como alcance a todo el sistema.
  - Producidos por el mismo sistema (por ejemplo, batería baja)
  - Desde otras aplicaciones (alertar que ocurrió un evento, por ejemplo)

# BroadcastReceivers

- Responden a anuncios que tienen como alcance a todo el sistema.
  - Producidos por el mismo sistema (por ejemplo, batería baja)
  - Desde otras aplicaciones (alertar que ocurrió un evento, por ejemplo)
- Empiezan y terminan su trabajo dentro del llamado del callback **onReceive**, por lo que tampoco debería mantenerse ningún Wakelock al terminar.

# ContentProviders

- Dan una forma de encapsular un set de datos estructurado.

# ContentProviders

- Dan una forma de encapsular un set de datos estructurado.
- Cada callback es típicamente una unidad de trabajo, por lo que todos los locks deben ser liberados al finalizar el llamado.

# Bonus: Intent-based Component Comunication

- Los componentes se pueden comunicar via mensajes asincrónicos llamdos **Intents**, que ofrecen una conexión entre componentes de la misma o de distintas aplicaciones.

# Bonus: Intent-based Component Comunication

- Los componentes se pueden comunicar via mensajes asincrónicos llamdos **Intents**, que ofrecen una conexión entre componentes de la misma o de distintas aplicaciones.
- Pueden ser explícitos (apuntando a un componente por su nombre) o implícitos (apuntando a una acción a realizar).

# Bonus: Intent-based Component Comunication

- Los componentes se pueden comunicar via mensajes asincrónicos llamdos **Intents**, que ofrecen una conexión entre componentes de la misma o de distintas aplicaciones.
- Pueden ser explícitos (apuntando a un componente por su nombre) o implícitos (apuntando a una acción a realizar).
- Es útil contabilizar de manera precisa la comunicación entre componentes, ya que es común que se adquieran Wakelocks en un componente y se liberen con otro invocado de forma asincrónica con un intent.

# Políticas de energía propuestas en el paper

El objetivo del mismo es **mostrar la ausencia de casos** en los que se adquiere un Wakelock pero el mismo no se libera en el punto apropiado del ciclo de vida de la aplicación. Para eso se definen ciertas políticas de manejo de energía, sobre las cuales se analizan las aplicaciones probadas.

# Definiciones

**Energy States:** Se definen los estados **high** y **low energy state**, cuando el dispositivo está y no está sosteniendo un Wakelock, respectivamente. Las políticas definidas verifican si en ciertos puntos de salida claves, el componente de software está en un low energy state (es decir, *habiendo liberado todos los Wakelocks*) como debería, por haber terminado su trabajo.

# Definiciones

**Energy States:** Se definen los estados **high** y **low energy state**, cuando el dispositivo está y no está sosteniendo un Wakelock, respectivamente. Las políticas definidas verifican si en ciertos puntos de salida claves, el componente de software está en un low energy state (es decir, *habiendo liberado todos los Wakelocks*) como debería, por haber terminado su trabajo.

**Exit Points:** Se necesita identificar el **punto de salida** de cada componente, en el cual debe estar en un *low energy state* al llamar el respectivo callback.

# Definiciones

**Component Policies:** Para identificar los *exit points* se particionaron los componentes en categorías, para las cuales se identificaron los callbacks en los que dichos *exit points* deben estar en un *low energy state*. Los componentes que no tienen un ciclo de vida bien definido o que no están comprendidos en el análisis, son tratados de forma conservativa, requiriendo que *todos* sus callbacks estén en un *low energy state*.

# Definiciones

**Component Policies:** Para identificar los *exit points* se particionaron los componentes en categorías, para las cuales se identificaron los callbacks en los que dichos *exit points* deben estar en un *low energy state*. Los componentes que no tienen un ciclo de vida bien definido o que no están comprendidos en el análisis, son tratados de forma conservativa, requiriendo que *todos* sus callbacks estén en un *low energy state*.

**Asynchrony:** Empíricamente una de las situaciones que más se repitió es la de un componente pidiendo un Wakelock y llamando a otro, mediante un *intent*, que luego lo libera. El primero está en un *high energy state*, pero el programa es *energy-safe* siempre y cuando el componente llamado se encargue de la liberación del lock.

# Dataflow Facts

Se utilizan los **Dataflow Facts** para monitorear el *energy state* en cada punto del programa. Lo representan con un **conjunto de Wakelocks adquiridos**, y catalogan al programa como en *low energy state* cuando dicho conjunto está **vacío**.

# Asynchronous Calls

Las **llamadas asincrónicas** son lanzadas por los *intents*. Para identificar el objetivo de la llamada se usa un procedimiento *def-use* en el que se registran las definiciones de los parámetros para inferir el componente que está siendo llamado. Sólamente se manejan los intents explícitos con un único objetivo especificado.

# Asynchronous Calls

Las **llamadas asincrónicas** son lanzadas por los *intents*. Para identificar el objetivo de la llamada se usa un procedimiento *def-use* en el que se registran las definiciones de los parámetros para inferir el componente que está siendo llamado. Sólamente se manejan los intents explícitos con un único objetivo especificado.

Cada componente llama a sus callbacks según el *lifecycle protocol*, que especifica en que órden se hace. El protocolo se representa con una **máquina de estado** cuyos *vértices* son los *callbacks* y los *ejes* denotan al *sucesor del callback*.

# Dataflow Analysis

El **análisis del Dataflow** se hace computando el *conjunto de Wakelocks* en cada paso del programa. Los lock facts son *generados* en los pasos **Wakelock.acquire**, *propagados sin cambios* en los ejes normales y *eliminados* al llamarse **Wakelock.release**. En caso de haber un *punto de encuentro entre caminos*, el conjunto de Wakelocks es representado por la **unión de los conjuntos en los puntos anteriores**.

# Dataflow Analysis

El **análisis del Dataflow** se hace computando el *conjunto de Wakelocks* en cada paso del programa. Los lock facts son *generados* en los pasos **Wakelock.acquire**, *propagados sin cambios* en los ejes normales y *eliminados* al llamarse **Wakelock.release**. En caso de haber un *punto de encuentro entre caminos*, el conjunto de Wakelocks es representado por la **unión de los conjuntos en los puntos anteriores**.

Una vez computado dicho análisis, se determina la *categoría* del componente. Luego se chequea si para dicha categoría, el mismo respeta que *en su exit point*, está en *low energy state* como corresponde, o no. Si lo está, **se lo verifica como energy safe**, si no, se levanta un **warning flag**.

# Experimentación

¿Con cuántas aplicaciones se experimentó?

# Experimentación

¿Con cuántas aplicaciones se experimentó?

- Se descargaron **2718 aplicaciones**.

# Experimentación

¿Con cuántas aplicaciones se experimentó?

- Se descargaron **2718 aplicaciones**.
- De esas, se concentraron en las **740 (27.2 %)** que usaban la **Wakelock API**.

# Experimentación

¿Con cuántas aplicaciones se experimentó?

- Se descargaron **2718 aplicaciones**.
- De esas, se concentraron en las **740 (27.2 %)** que usaban la **Wakelock API**.
- Se extrajo el contenido de los archivos **.apk (Android Package)** y se convirtió a *bytecode de Java*.

# Experimentación

¿Con cuántas aplicaciones se experimentó?

- Se descargaron **2718 aplicaciones**.
- De esas, se concentraron en las **740 (27.2 %)** que usaban la **Wakelock API**.
- Se extrajo el contenido de los archivos **.apk (Android Package)** y se convirtió a *bytecode de Java*.
- Por las *limitaciones* de las herramientas usadas, solo **328 aplicaciones (44.3 %)** se pudieron convertir efectivamente.

# Experimentación

¿Con cuántas aplicaciones se experimentó?

- Se descargaron **2718 aplicaciones**.
- De esas, se concentraron en las **740 (27.2 %)** que usaban la **Wakelock API**.
- Se extrajo el contenido de los archivos **.apk (Android Package)** y se convirtió a *bytecode de Java*.
- Por las *limitaciones* de las herramientas usadas, solo **328 aplicaciones (44.3 %)** se pudieron convertir efectivamente.

¿Y qué pasó con esas 328 aplicaciones?

# Experimentación

¿Con cuántas aplicaciones se experimentó?

- Se descargaron **2718 aplicaciones**.
- De esas, se concentraron en las **740 (27.2 %)** que usaban la **Wakelock API**.
- Se extrajo el contenido de los archivos **.apk (Android Package)** y se convirtió a *bytecode de Java*.
- Por las *limitaciones* de las herramientas usadas, solo **328 aplicaciones (44.3 %)** se pudieron convertir efectivamente.

¿Y qué pasó con esas 328 aplicaciones?

- Se encontró que **145 (44.2 %)** cumplían totalmente con las políticas antes descriptas.

# Experimentación

¿Con cuántas aplicaciones se experimentó?

- Se descargaron **2718 aplicaciones**.
- De esas, se concentraron en las **740 (27.2 %)** que usaban la **Wakelock API**.
- Se extrajo el contenido de los archivos **.apk (Android Package)** y se convirtió a *bytecode de Java*.
- Por las *limitaciones* de las herramientas usadas, solo **328 aplicaciones (44.3 %)** se pudieron convertir efectivamente.

¿Y qué pasó con esas 328 aplicaciones?

- Se encontró que **145 (44.2 %)** cumplían totalmente con las políticas antes descriptas.
- En las **183 (55.8 %)** restantes no se pudo mostrar que las políticas se cumplieran (lo que **no quiere decir que no se hayan cumplido**).

# Experimentación

¿Con cuántas aplicaciones se experimentó?

- Se descargaron **2718 aplicaciones**.
- De esas, se concentraron en las **740 (27.2 %)** que usaban la **Wakelock API**.
- Se extrajo el contenido de los archivos **.apk (Android Package)** y se convirtió a *bytecode de Java*.
- Por las *limitaciones* de las herramientas usadas, solo **328 aplicaciones (44.3 %)** se pudieron convertir efectivamente.

¿Y qué pasó con esas 328 aplicaciones?

- Se encontró que **145 (44.2 %)** cumplían totalmente con las políticas antes descriptas.
- En las **183 (55.8 %)** restantes no se pudo mostrar que las políticas se cumplieran (lo que **no quiere decir que no se hayan cumplido**).
- Se **inspeccionaron manualmente 50 de esas aplicaciones**, elegidas aleatoriamente, y *se encontraron bugs* e *imprecisiones en el análisis planteado*.

# Bugs comunes

Los errores más comunes ocurren en la implementación de **activities** con Wakelocks.

# Bugs comunes

Los errores más comunes ocurren en la implementación de **activities** con Wakelocks.

Los desarrolladores **suelen olvidar** liberar los recursos cuando implementan los callbacks **onPause** y **onStop**, haciendo que **el recurso pedido se mantenga encendido** incluso cuando el usuario navegó **fuera de dicha activity**, ya sea **activando otra activity** o **presionando el botón Home**.

# Bugs comunes

Los errores más comunes ocurren en la implementación de **activities** con Wakelocks.

Los desarrolladores **suelen olvidar** liberar los recursos cuando implementan los callbacks **onPause** y **onStop**, haciendo que **el recurso pedido se mantenga encendido** incluso cuando el usuario navegó **fuera de dicha activity**, ya sea **activando otra activity** o **presionando el botón Home**.

Varias aplicaciones, además, violaron la política de **BroadcastReceivers**: este objeto es únicamente válido durante la ejecución de un **onReceive**.

# Bugs comunes

Los errores más comunes ocurren en la implementación de **activities** con Wakelocks.

Los desarrolladores **suelen olvidar** liberar los recursos cuando implementan los callbacks **onPause** y **onStop**, haciendo que **el recurso pedido se mantenga encendido** incluso cuando el usuario navegó **fuera de dicha activity**, ya sea **activando otra activity** o **presionando el botón Home**.

Varias aplicaciones, además, violaron la política de **BroadcastReceivers**: este objeto es únicamente válido durante la ejecución de un **onReceive**.

La herramienta **levantó un flag** al mantenerse un Wakelock luego de dicho callback.

# Motivos de imprecisión en el análisis

Como mencionamos antes, que la herramienta diga que una app no verifica las políticas definidas **no implica que la misma tenga bugs de energía**. Esto es *inherente al análisis estático* y lleva a **falsos positivos** (es decir, la aplicación es correcta pero la herramienta no puede decirlo).

# Motivos de imprecisión en el análisis

Como mencionamos antes, que la herramienta diga que una app no verifica las políticas definidas **no implica que la misma tenga bugs de energía**. Esto es *inherente al análisis estático* y lleva a **falsos positivos** (es decir, la aplicación es correcta pero la herramienta no puede decirlo).

Ejemplos de esto es son los casos en los que *el lock se adquiere y se libera con la misma condición*, o cuando *el lock es adquirido/liberado bajo alguna condición muy compleja*.

# Conclusiones

La contribución principal de este paper es **la herramienta** para verificar si cierta aplicación de Android cumple o no con el conjunto de políticas respecto al uso de la **Wakelock API**. Este análisis **permitió definir varios bugs comunes** y ayuda a **entender el uso de recursos** en aplicaciones del mundo real, para intentar evitar los bugs relacionados a su uso en futuros desarrollos.