모바일 시스템 프로그래밍

10 Energy-efficient Mobile Sensing System Design 2

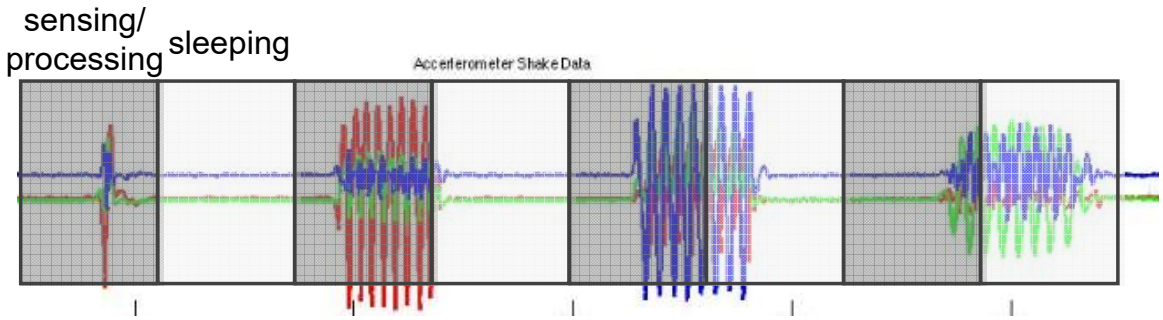
2017 1학기

강승우

Duty Cycling Implementation

- 안드로이드 플랫폼 기반으로 모바일 센싱을 할 때 Duty Cycling을 어떻게 구현할 수 있을까?

- Duty Cycling을 이용하여 energy efficiency를 높인다고 할 때 Active state와 Inactive(sleep) state를 반복해서 변경할 수 있어야 함
  - Active state: sensing, processing을 수행하는 구간
  - Inactive(sleep) state: sensor off, CPU off 구간



Duty Cycling Implementation

- 지금까지 주기적인 연산 처리를 위해 사용했던 방식
  - Thread 혹은 Timer/TimerTask를 이용해서 주기적인 연산을 수행하였음

  → 그러나 이 방법은 CPU가 ON인 상태에서만 동작할 수 있었음

  → 스마트폰 화면이 꺼진 후 sleep 상태에 들어가면 동작하지 않음

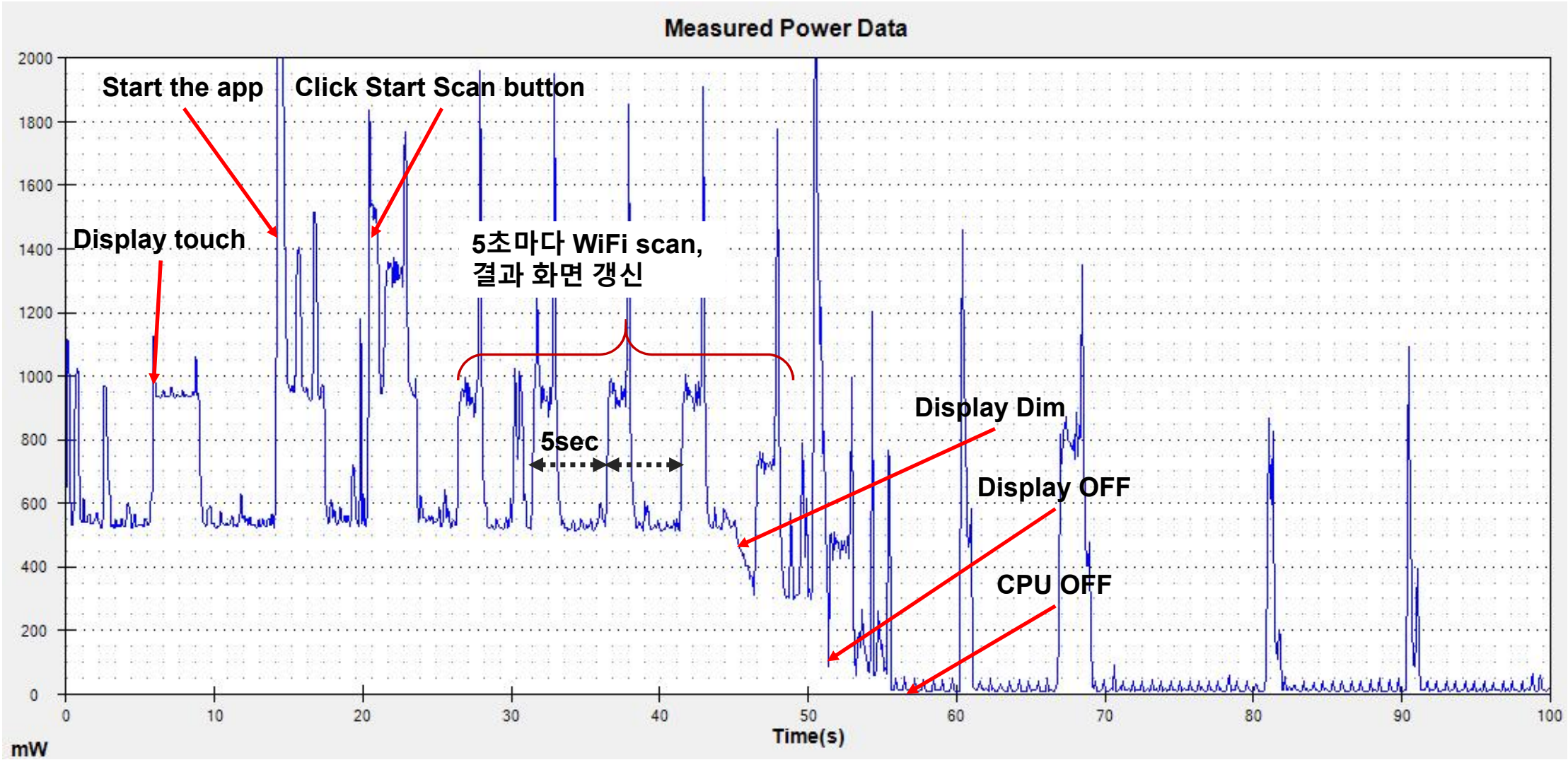
  - Wakelock을 이용하여 CPU가 sleep 상태에 들어가지 않도록 하여 화면이 꺼지더라도 주기적으로 백그라운드에서 연산이 되도록 하였음

  → 즉, 센서는 duty cycling을 했다고 볼 수 있지만, CPU는 100% duty cycle를 적용한 것임 (실질적으로는 duty cycling을 하지 않은 것)

  - 예제: MSP10WifiScanTimer
    - 버전 1, 버전 2: 동작의 차이를 확인하기 위해 Power Monitor 출력 결과를 다시 보자

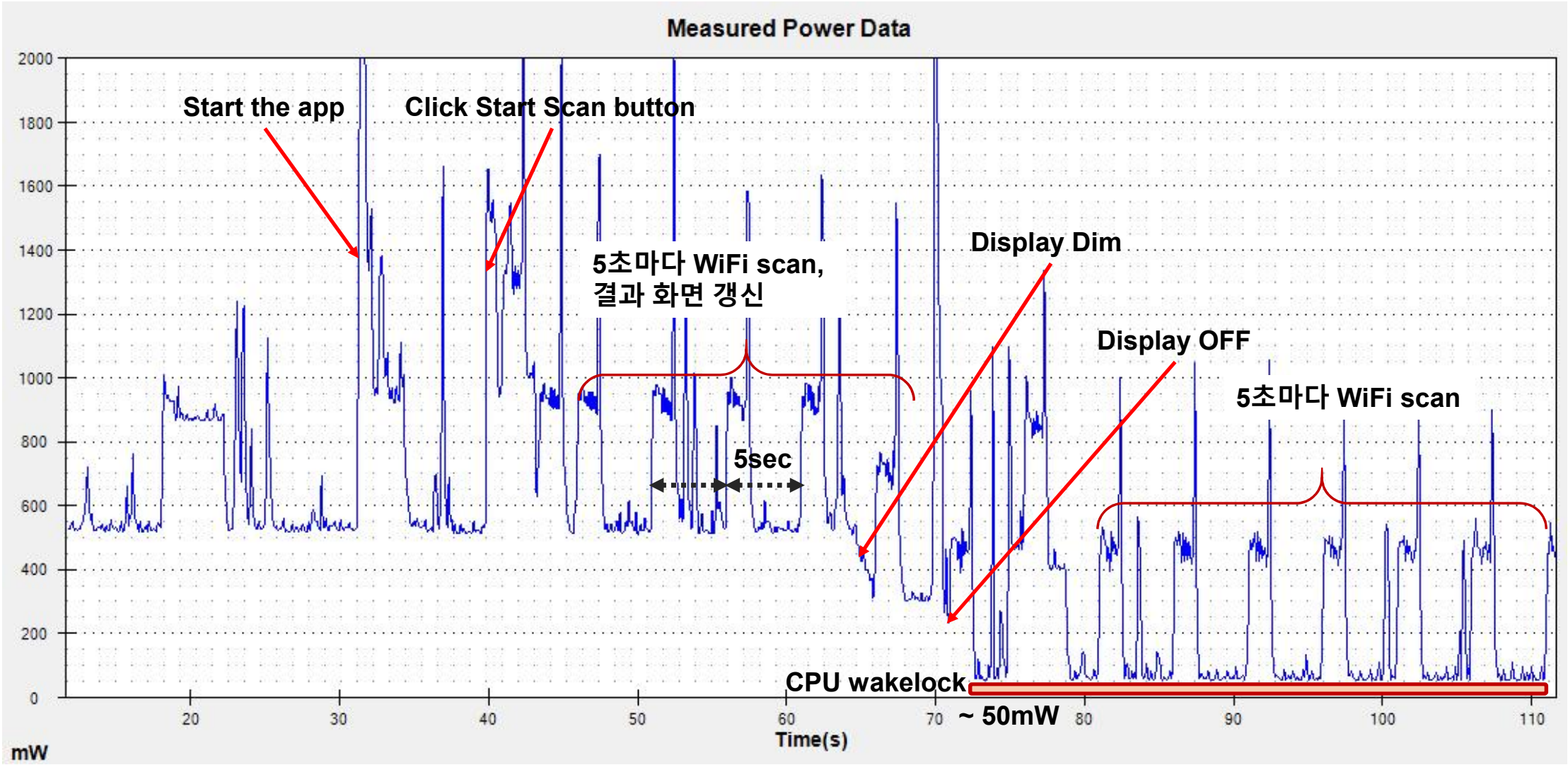
Avg. Power (버전 1: wakelock 미사용)

(WiFi ON, GPS / BT OFF 상태)



Avg. Power (버전 2: wakelock 사용)

(WiFi ON, GPS / BT OFF 상태)



Duty Cycling Implementation

- 문제
  - Wakelock을 사용하지 않으면, CPU가 OFF가 되므로 코드가 실행되지 않음 (주기적인 센싱/연산을 수행할 수가 없음)
  - Wakelock을 사용하면, CPU가 항상 ON이 되므로 에너지 낭비가 있음

- 필요할 때만 주기적으로 CPU를 사용할 수 있는 방법이 필요
  - 안드로이드 애플리케이션 코드가 실행 중이지 않더라도
    혹은 디바이스가 sleep 상태에 있더라도
    애플리케이션이 정해진 시간/간격으로 실행될 수 있도록 하는 방법이 필요

Duty Cycling Implementation

- AlarmManager 클래스
  - 애플리케이션이 실행 중이지 않은 경우에도, 미래의 특정 시간에 애플리케이션을 실행할 수 있도록 해 줌
  - 대표적인 애플리케이션: 알람 시계 (특정 시각(예: 오전 8시)에 혹은 특정 시간 후(예: 1시간 후) 알람을 울림)
- AlarmManager를 통한 Alarm Service 이용
  - 이것도 역시 안드로이드 시스템 서비스임
  - AlarmManager 클래스에 정의된 API를 이용하여,
    애플리케이션 코드를 실행하려고 하는 타이밍에 맞춰서,
    안드로이드 시스템에 알람을 발생하도록 요청할 수 있음

https://developer.android.com/reference/android/app/AlarmManager.html?hl=ko
https://developer.android.com/training/scheduling/alarms.html?hl=ko

Duty Cycling with AlarmManager

- Alarm Service 특징
  - 설정한 시각 혹은 일정 주기마다 필요한 Intent를 실행할 수 있게 해준다
  - 다른 필요한 연산을 수행할 수 있게 서비스를 시작하는 Broadcast Receiver와 함께 사용할 수 있다
  - 애플리케이션이 실행 중이지 않을 때나 디바이스가 sleep 상태에 있을 때도 애플리케이션에서 어떤 액션을 수행할 수 있게 해준다
  - 애플리케이션의 자원 사용량을 최소화할 수 있도록 도와준다
    - 그러나 Alarm을 잘못 사용하면 불필요한 자원 낭비가 발생할 수 있음
      - Wakelock과 마찬가지로 애플리케이션에서 AlarmManager를 이용하여 alarm 요청을 무분별하게 할 경우 Power Management가 제대로 안 될 수 있음

Duty Cycling with AlarmManager

- Alarm Service의 주된 사용 목적
  - 애플리케이션이 실행 중이지 않거나, 디바이스가 sleep 상태인 경우에도 미래 특정 시점에 어떤 작업을 수행해야 할 경우

  ✓일반적인 애플리케이션의 실행 시간 동안에 일어날 것이 확실한 timing 연산 (one time or periodic)의 경우는 Handler 클래스를 Timer나 Thread와 함께 사용하여 처리하는 것이 좋다

Duty Cycling with AlarmManager

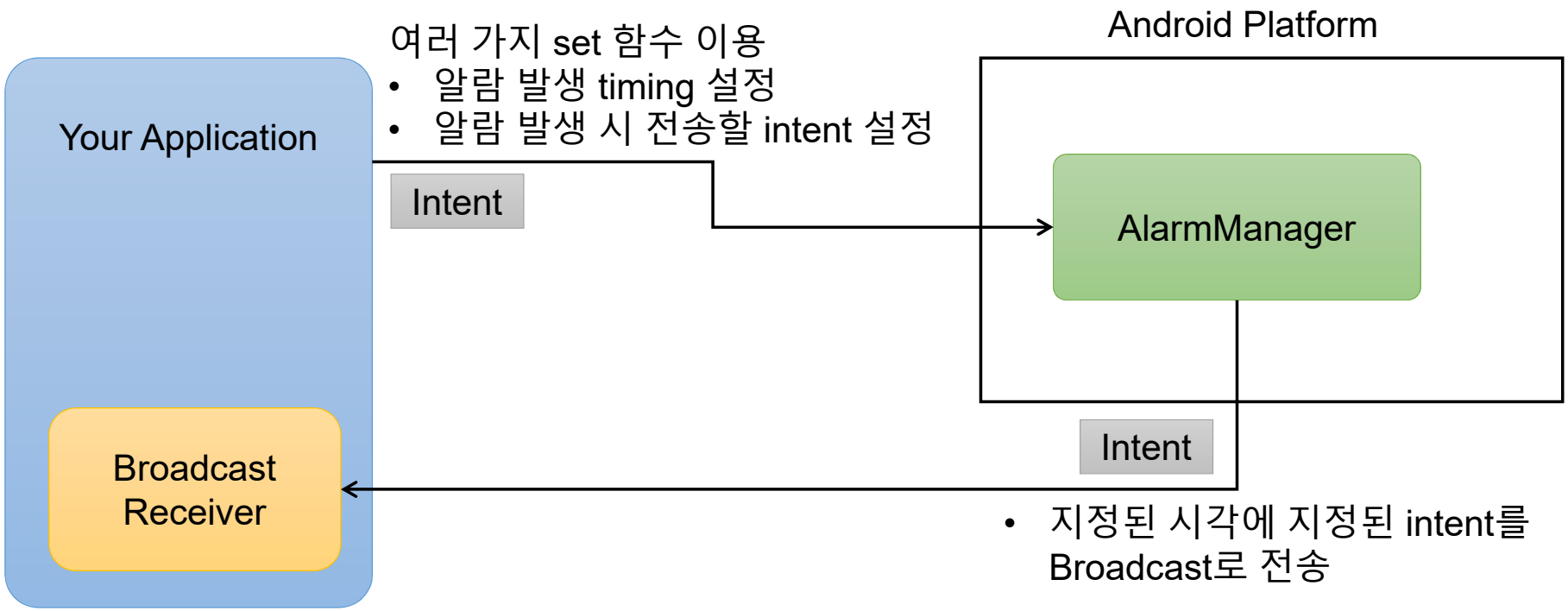
- AlarmManager 사용 예제

```
AlarmManager am;
PendingIntent pendingIntent;
.....

am = (AlarmManager)getSystemService(ALARM_SERVICE);

// Alarm 발생 시 전송되는 broadcast를 수신할 receiver 등록
IntentFilter intentFilter = new IntentFilter("kr.ac.koreatech.msp.alarm");
registerReceiver(AlarmReceiver, intentFilter);

// Alarm이 발생할 시간이 되었을 때, 안드로이드 시스템에 전송을 요청할 broadcast를 지정
Intent intent = new Intent("kr.ac.koreatech.msp.alarm");
pendingIntent = PendingIntent.getBroadcast(this, 0, intent, 0);

// Alarm이 발생할 시간 및 alarm 발생시 이용할 pending intent 설정
// 5초 후 alarm 발생
am.setExact(AlarmManager.ELAPSED_REALTIME_WAKEUP,
            SystemClock.elapsedRealtime() + 5000,
            pendingIntent);
```

```
// AlarmManager에 등록한 alarm 취소
am.cancel(pendingIntent);
```

Duty Cycling with AlarmManager

- AlarmManager를 이용한 애플리케이션 동작



Duty Cycling with AlarmManager

- Alarm 발생 설정
  - 시간 기준
    - 부팅 후 경과 시간(elapsed time) vs. 실제 시간 (RTC: real time clock)
    - ELAPSED_REALTIME / RTC
  - Sleep 상태에서 수행 vs. 수행하지 않음
    - 디바이스가 sleep 상태에 있으면 wake up을 하고 작업 수행 혹은 sleep 상태에서는 수행하지 않음
  - 일회성 알람 vs. 반복 알람
    - AlarmManager의 alarm 설정 메소드를 이용하여 지정
    - set() / setRepeating()
  - 정확한 시각에 수행 vs. 오차 허용
    - setExact() / setInexactRepeating()

https://developer.android.com/reference/android/app/AlarmManager.html?hl=ko

Duty Cycling with AlarmManager

- 시간 기준과 sleep 상태에서 수행 여부는 AlarmManager에 정의된 4가지 상수를 이용하여 지정
  - AlarmManager.ELAPSED_REALTIME
    - 디바이스가 부팅된 이후 경과된 시간을 기준으로 함

    → 예를 들어 현재부터 몇 시간 후에 alarm 발생 혹은 몇 시간 후에 얼마 간격으로 발생
  - AlarmManager.ELAPSED_REALTIME_WAKEUP
    - ELAPSED_REALTIME과 동일하며, sleep 상태일 경우 디바이스를 활성 상태로 전환한 후 작업을 수행함

    → 시스템 내부적으로 wakelock을 잡는 것임 (앱에서 wakelock을 요청하는 것이 아니라)
  - AlarmManager.RTC
    - 실제 시각을 기준으로 함

    → 국제 표준시 기반으로 월, 일, 시, 분을 정확히 지정하는 경우
  - AlarmManager.RTC_WAKEUP
    - RTC와 동일하며, sleep 상태일 경우 디바이스를 활성 상태로 전환한 후 작업을 수행함

Duty Cycling with AlarmManager

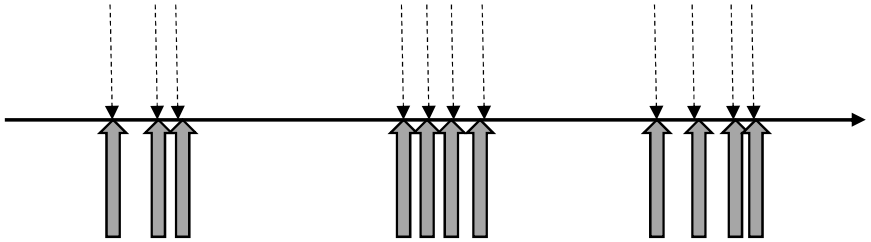
- 반복 여부, 오차 허용 여부는 서로 다른 set 함수를 이용하여 설정
  - public void set (int type, long triggerAtTime, PendingIntent operation)
    - 한 번만 수행되는 알람을 예약
  - public void setRepeating (int type, long triggerAtTime, long interval, PendingIntent operation)
    - 주기적으로 반복 수행되는 알람을 예약
  - public void setInexactRepeating (int type, long triggerAtTime, long interval, PendingIntent operation)
    - setRepeating() 메서드와 동일. 하지만 알람이 지정된 시각에 정확하게 일어나지 않을 수 있음. 정확성보다는 일정 주기로 특정 작업을 수행하기만 하면 될 때 사용
  - public void setExact (int type, long triggerAtMillis, PendingIntent operation)
    - Added in API level 19 (KitKat)
    - 지정한 시간에 정확히 일어나는 알람을 예약
    - set() 메서드와 동일. OS가 alarm 발생 시간을 조절하는 것을 불허

Duty Cycling with AlarmManager

- API level 19부터는 set/setRepeating 모두 inexact로 변경됨
  - 즉 안드로이드 OS가 alarm 발생 시간을 조정할 수 있음
  - 여러 개의 애플리케이션에서 알람 발생 요청이 있을 때, 그것들을 모아서 비슷한 시간대에 있는 알람은 그 타이밍에 일괄적으로 발생

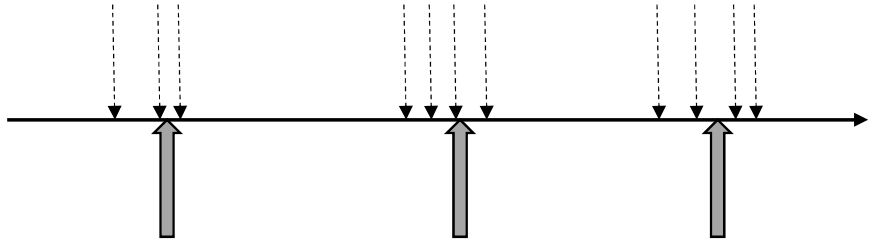
  → 자원 사용에 미치는 영향을 줄이기 위함

Alarm 요청 (모두 wakeup alarm이라고 가정하자)



- Sleep 상태에 있을 때 wakeup 되는 횟수와 시간이 증가하여 에너지 소모가 많아질 수 있음

Alarm 요청 (모두 wakeup alarm이라고 가정하자)



Alarm 발생으로 인해 wakelock이 잡히는 시점

Duty Cycling with AlarmManager

- 그럼 정확한 timing으로 반복되는 alarm은 어떻게 구현?
  - setExact를 반복 사용하는 방식으로 구현

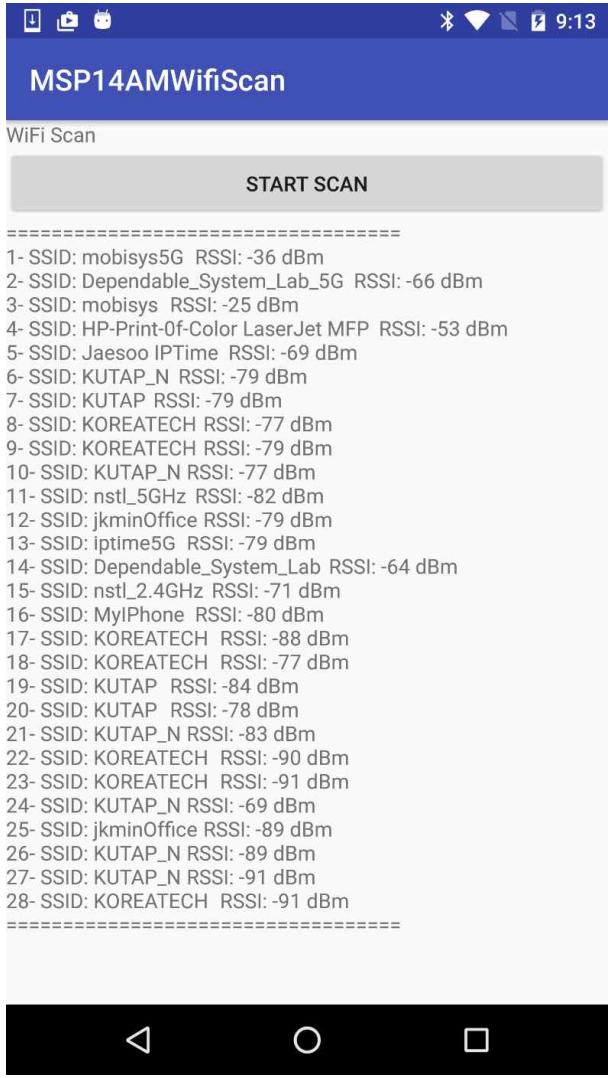
  - setExact 함수로 일회성 알람을 예약하고,
    알람 발생시 다시 setExact 함수를 이용하여 동일한 조건의 알람을 예약

Duty Cycling with AlarmManager

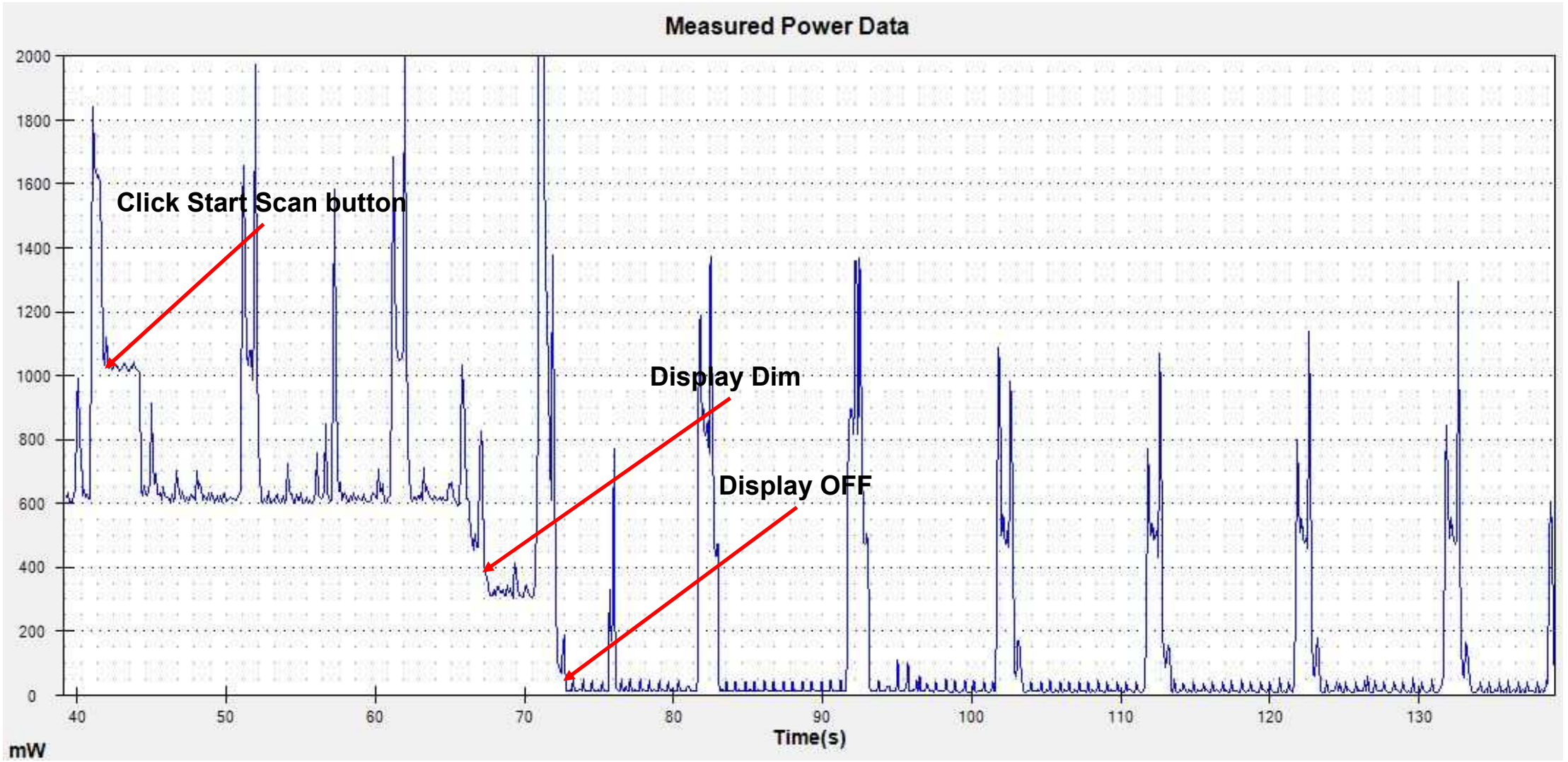
- void setExact (int type, long triggerAtMillis, PendingIntent operation)
  - type
    - 알람의 속성을 지정
    - AlarmManager.ELAPSED_REALTIME / AlarmManager.ELAPSED_REALTIME_WAKEUP
    - AlarmManager.RTC / AlarmManager.RTC_WAKEUP
  - triggerAtMillis
    - 알람 type(ELAPSED_REALTIME vs. RTC)에 따라 지정하는 방식이 다름
    - ELAPSED_REALTIME를 이용하는 경우 SystemClock.elapsedRealtime() + 원하는 시간 (밀리초 단위의 시간. 즉, 10초이면 10*1000)
    - RTC를 이용하는 경우 System.currentTimeMillis() 사용
      - 예제: https://developer.android.com/training/scheduling/alarms.html?hl=ko 참고할 것
  - operation
    - 실행할 작업을 pendingIntent 객체로 지정

Duty Cycling with AlarmManager

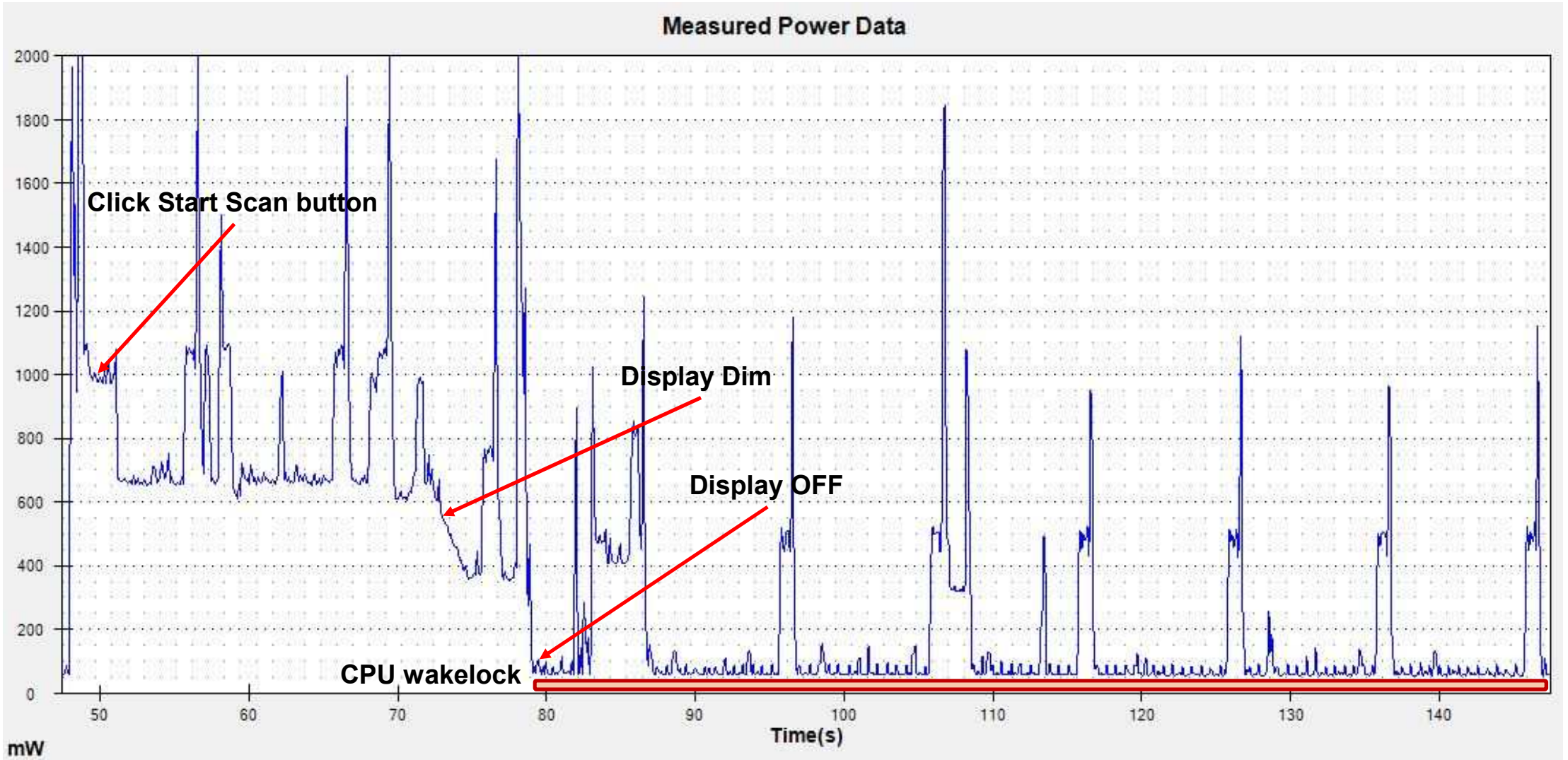
- 예제
  - Wakelock을 사용하지 않고 sleep 상태에서도 주기적으로 WiFi 스캔을 실행할 수 있게 만들어 보자
  - 프로젝트 이름: MSP14AMWifiScan
    - 관련 예제 프로젝트: MSP10WifiScanTimer, MSP06IndoorProximityAlert
  - Start scan 버튼을 누르면 일정 시간 간격으로 wifi scan 수행
  - AlarmManager를 이용하여 sleep 상태에서도 wakeup하여 wifi scan을 수행하도록 함



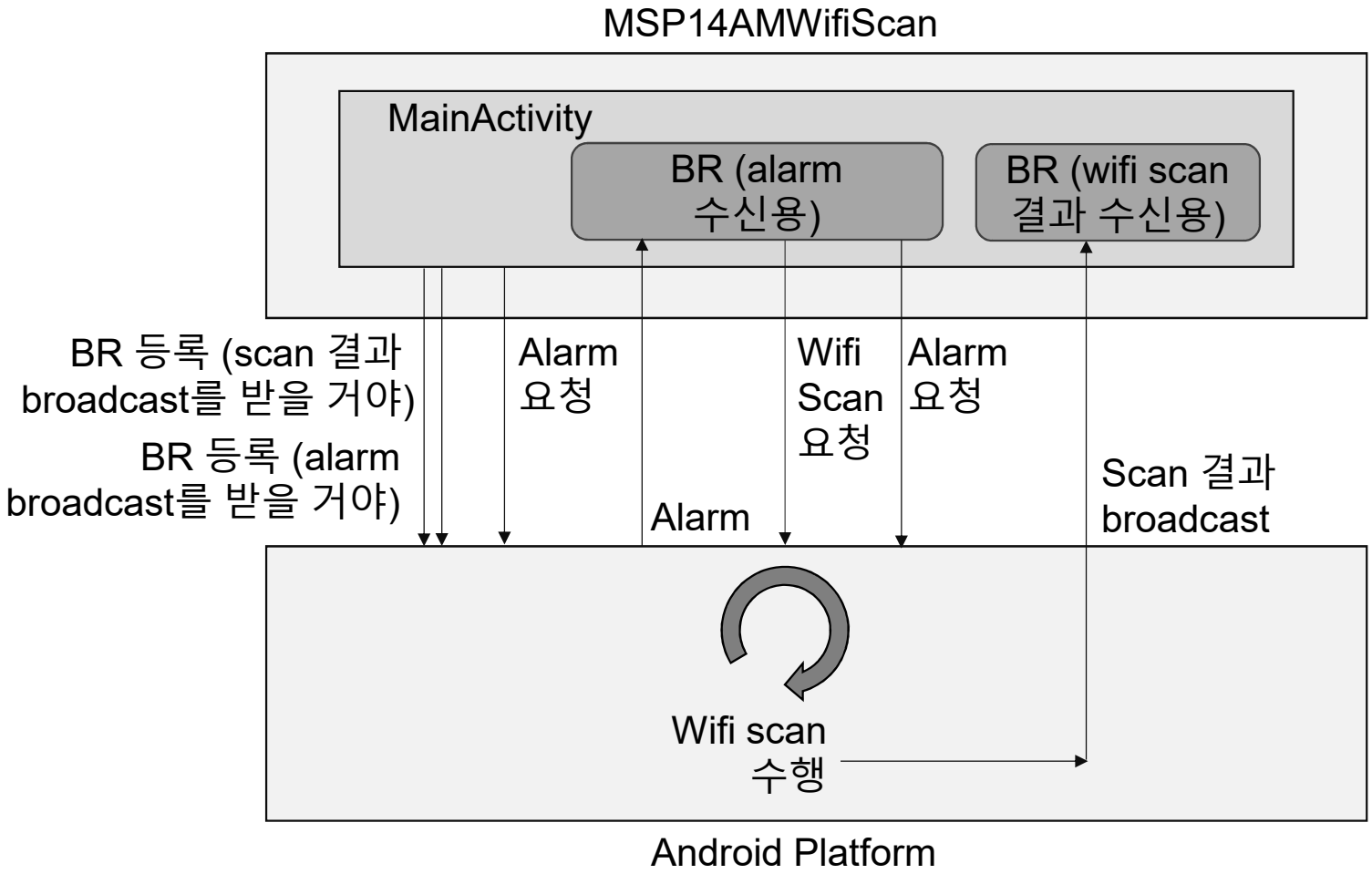
Power Monitor로 동작 확인 (MSP14AMWifiScan: AlarmManager 사용)



Power Monitor로 동작 확인 (MSP10WifiScanTimer: Wakelock 사용)



예제 애플리케이션 구조


MSP14AMWifiScan
MainActivity
BR (alarm 수신용)
BR (wifi scan 결과 수신용)
BR 등록 (scan 결과 broadcast를 받을 거야)
BR 등록 (alarm broadcast를 받을 거야)
Alarm 요청
Alarm
Wifi Scan 요청
Alarm 요청
Scan 결과 broadcast
Wifi scan 수행
Android Platform

Code snippet (MainActivity.java)

```
@Override
protected void onCreate(Bundle savedInstanceState) {
    super.onCreate(savedInstanceState);
    setContentView(R.layout.activity_main);

    scanResultText = (TextView)findViewById(R.id.result);

    wifiManager = (WifiManager)getSystemService(WIFI_SERVICE);
    if(wifiManager.isWifiEnabled() == false)
        wifiManager.setWifiEnabled(true);


    am = (AlarmManager)getSystemService(ALARM_SERVICE);
}
```

```
public void onClick(View view) {
    if(view.getId() == R.id.start) {
        //Toast.makeText(this, "WiFi scan start!!", Toast.LENGTH_LONG).show();
        startScan();
    }
}
private void startScan() {
    // wifi scan 결과 발생 시 전송되는 broadcast를 수신할 receiver 등록
    IntentFilter filter = new IntentFilter(WifiManager.SCAN_RESULTS_AVAILABLE_ACTION);
    registerReceiver(mReceiver, filter);

    // Alarm 발생 시 전송되는 broadcast를 수신할 receiver 등록
    IntentFilter intentFilter = new IntentFilter("kr.ac.koreatech.msp.alarm");
    registerReceiver(AlarmReceiver, intentFilter);

    // Alarm이 발생할 시간이 되었을 때, 안드로이드 시스템에 전송을 요청할 broadcast를 지정
    Intent intent = new Intent("kr.ac.koreatech.msp.alarm");
    pendingIntent = PendingIntent.getBroadcast(this, 0, intent, 0);

    // Alarm이 발생할 시간 및 alarm 발생시 이용할 pending intent 설정
    // 10초 후 alarm 발생
    am.setExact(AlarmManager.ELAPSED_REALTIME_WAKEUP,
            SystemClock.elapsedRealtime() + 10000, pendingIntent);
}
```

```
// Alarm 시간이 되었을 때 안드로이드 시스템이 전송해주는 broadcast를 받을 receiver 정의
// 그리고 다시 동일 시간 후 alarm이 발생하도록 설정한다.
private BroadcastReceiver AlarmReceiver = new BroadcastReceiver() {
    @Override
    public void onReceive(Context context, Intent intent) {
        if(intent.getAction().equals("kr.ac.koreatech.msp.alarm")) {

            //*****************
            // Alarm이 발생하였을 때 wifi scan을 수행한다
            wifiManager.startScan();
            scanResultText.setText("");
            //---------------
            // Alarm receiver에서는 장시간에 걸친 연산을 수행하지 않도록 한다
            // Alarm을 발생할 때 안드로이드 시스템에서 wakelock을 잡기 때문에 CPU를 사용할 수 있지만
            // 그 시간은 제한적이기 때문에 애플리케이션에서 필요하면 wakelock을 잡아서 연산을 수행해야 함
            //*****************

            Intent in = new Intent("kr.ac.koreatech.msp.alarm");
            pendingIntent = PendingIntent.getBroadcast(MainActivity.this, 0, in, 0);
            am.setExact(AlarmManager.ELAPSED_REALTIME_WAKEUP,
                    SystemClock.elapsedRealtime() + 10000, pendingIntent);
        }
    }
};
```

WiFi scan 결과를 받는 Broadcast Receiver는 기존 코드와 동일

```
@Override
protected void onDestroy() {
    super.onDestroy();

    unregisterReceiver(mReceiver);

    try {
        // Alarm 발생 시 전송되는 broadcast 수신 receiver를 해제
       unregisterReceiver(AlarmReceiver);
    } catch(IllegalArgumentException ex) {
        ex.printStackTrace();
    }
    // AlarmManager에 등록한 alarm 취소
       am.cancel(pendingIntent);
}
```

Duty Cycling with AlarmManager

- 안드로이드 시스템이 알람을 발생하여 애플리케이션에 Broadcast로 알려줄 때 Broadcast Receiver의 onReceive() 메소드가 실행됨
- onReceive() 메소드가 실행되는 동안에는 AlarmManager가 CPU wakelock을 잡고 있어서 필요한 CPU 연산 수행 가능
  - onReceive()메소드가 실행 완료되면 AlarmManager는 wakelock을 해제하게 됨 → 즉 디바이스가 다시 sleep 상태로 들어가게 됨
- 만약 Broadcast Receiver 에서 startService()로 백그라운드 서비스를 실행한다면, 서비스가 실제 시작되기 전에 디바이스가 sleep 상태가 되어버릴 수도 있음
- 이런 문제를 방지하기 위해서는 Broadcast Receiver와 Service에서 필요한 경우 별도의 wakelock 사용/해제 처리를 해주어야 함

Duty Cycling with AlarmManager

- Alarm 시간 설정 제약 (Android 5. 대 버전 이후)
  - Alarm 발생 시간이 5초 이내인 경우 안드로이드 시스템에서 일괄적으로 5초로 설정
  - 즉, 아래와 같이 1초 후 alarm 발생 요청을 하더라도 실제 alarm은 5초 후에 발생함

```
am.setExact(AlarmManager.ELAPSED_REALTIME_WAKEUP,
        SystemClock.elapsedRealtime() + 1000, pendingIntent);
```

  - 앞의 예제에서 setExact 함수를 찾아 alarm 발생 시간을 5000 이하로 변경하여 테스트 해볼 것