2 Fundamentos teóricos

El objetivo de este capítulo es situar el desarrollo del proyecto bajo un contexto teórico que provea las bases para el desarrollo práctico del mismo. Se recogen a continuación los conceptos y protocolos que sustentan (ya sea directa o indirectamente, como en el caso de las redes mesh) el sistema a desarrollar.

2.1 Redes Mesh

Las redes *mesh*, o *de malla*, son sistemas multisalto (*multihop*) [3] en los cuales los nodos se ayudan el uno al otro para transmitir paquetes a través de la red, organizadas en una topología de malla. Este tipo de redes ad hoc se pueden poner en marcha con una preparación mínima, y proveen un sistema fiable y flexible que puede extenderse a miles de dispositivos, lo cual las hace especialmente interesantes en situaciones de emergencia. La fiabilidad, escalabilidad y adaptabilidad son las principales características de las redes mesh inalámbricas.

Las redes mesh son redes *peer-to-peer* (punto a punto), ad hoc y multisalto. Un nodo puede enviar y recibir mensajes, a la vez que hace la función de *router* (encaminador) y reenvía mensajes a sus nodos vecinos. A través del proceso de reenvío, un paquete de red encontrará una ruta hacia su destino, pasando por nodos intermedios con enlaces de comunicación fiables.

Al igual que Internet y otras redes punto a punto basadas en rutas, las redes de malla ofrecen múltiples rutas alternativas redundantes, de forma que si un enlace falla por cualquier razón, la red automáticamente encamina los mensajes a través de caminos alternativos. También, al reducir la distancia entre nodos, se aumenta la calidad del enlace, lo cual aumenta la fiabilidad de las comunicaciones sin un aumento de potencia en el transmisor. De esta forma, en una red mesh, aumentar el alcance, la redundancia y la fiabilidad general de la red es tan sencillo como añadir más nodos.

Una red mesh se organiza de manera automática, y no requiere de un administrador o configuración manual. De este modo, para desplazar un nodo o añadir uno nuevo, basta con simplemente encender el dispositivo. La red descubrirá el nuevo nodo y lo añadirá automáticamente al sistema. Esto hace que las redes mesh no sólo sean fiables, si no también adaptables: el hecho de perder uno o más nodos no debería afectar al

(a) Red de malla completa.

(b) Red de malla parcial.

Figura 2.1 Redes mesh.

funcionamiento de la red (siempre que la configuración de malla de la red no provoque una división en dos de la misma, o el aislamiento total de un nodo).

Acrónimo	Nombre	Funcionalidad
AHCP	Ad Hoc Configuration Protocol	Autoconfiguración
AODV	Ad hoc On-Demand Distance Vector	Encaminamiento
B.A.T.M.A.N	Better Approach To Mobile Adhoc Networking	Encaminamiento
DNVR	Dynamic Nix-Vector Routing	Encaminamiento
DSDV	Destination-Sequenced Distancce-Vector Routing	Encaminamiento
DSR	Dynamic Source Routing	Encaminamiento
DWCP	Dynamic WMN Configuration Protocol	Autoconfiguración
HSLS	Hazy-Sighted Link State	Encaminamiento
HWMP	Hybrid Wireless Mesh Protocol	Encaminamiento
OLSR	Optimized Link State Routing protocol	Encaminamiento
OSPF	Open Shortest Path First Routing	Encaminamiento
PAP	Proactive Autoconfiguration Protocol	Autoconfiguración
PWRP	Predictive Wireless Routing Protocol	Encaminamiento
TORA	Temporally-Ordered Routing Algorithm	Encaminamiento
ZRP	Zone Routing Protocol	Encaminamiento

Tabla 2.1 Algunos protocolos de encaminamiento y autoconfiguración de redes mesh.

Si bien existen numerosos protocolos de encaminamiento (**Tabla 2.1**), el IEEE ha desarrollado un conjunto de estándares bajo el título **802.11s** con el objetivo de definir la arquitectura y los protocolos de las redes mesh [5]. Se trata de una modificación del estándar **802.11** en la que se define cómo los dispositivos inalámbricos se han de comunicar para crear redes WLAN mesh, bien sea para topologías estáticas o redes ad hoc. Este nuevo estándar depende de los anteriores (802.11a, 802.11b, 802.11g o 802.11n) para soportar el tráfico [6]. Con respecto a los protocolos de encaminamiento, 802.11s establece el uso de HWMP. 802.11s también incluye mecanismos para proveer acceso determinista a la red, para control de la congestión y ahorro de energía.

En una red mesh no hay definidos roles: no hay clientes ni servidores. Los protocos de seguridad utilizados en una red mesh deben, por lo tanto, ser protocolos puramente punto a punto donde cualquiera de los extremos puede iniciar la comunicación con el otro. El estándar del IEEE define un protocolo seguro de autenticación basada en contraseña llamado SAE (*Simultaneous Authentication of Equals*, Autenticación simultánea de iguales). Este mecanismo está basado en el intercambio de claves Diffie-Hellman. Cuando los nodos se descubren (y hay seguridad establecida) toman parte en un intercambio SAE. Si este tiene éxito, cada nodo sabe que el otro lado posee la contraseña de la red, y establecen una clave criptográficamente robusta. Esta clave se utiliza con el AMPE (*Authenticated Mesh Peering Exchange*, Intercambio de emparejamiento mesh autenticado) para establecer un emparejamiento seguro y proteger el tráfico de la red.

2.2 IEEE 802.11

El estándar IEEE 802.11 fue definido en el año 1997 como nuevo estándar para las redes de área local inalámbrica, con el objetivo de sustituir a las capas física y de enlace del modelo OSI para redes cableadas (IEEE 802.3) especificando su funcionamiento en redes WLAN, haciendo que ambas redes sean idénticas excepto en la forma en la que los terminales acceden a la red.

La capa física de la especificación IEEE 802.11 se ocupa de definir los métodos por los que se difunde la señal. Ofrece 4 tipos de técnicas de transmisión:

- Infrarrojos: usa transmisión de corto alcance, con una velocidad de 1Mbps o 2Mbps.
- FHSS (*Frequency Hopping Spread Spectrum*, Espectro disperso por salto de frecuencia): se transmiten los datos saltando de canal a canal, de acuerdo a una secuencia de salto pseudo aleatoria particular que distribuye uniformemente la señal a través de la banda de frecuencia operativa. Una vez que la secuencia de saltos se configura en un AP, las estaciones se sincronizarán automáticamente según la secuencia de salto correcta. Se consiguen velocidades de transmisión de 1Mbps a 2Mbps. Opera en la banda de los 2,4GHz.

- DSSS (*Direct Sequence Spread Spectrum*, espectro disperso de secuencia directa): utiliza un rango de frecuencia amplio de 22 MHz todo el tiempo. La señal se expande a través de diferentes frecuencias. Cada bit de datos se convierte en una secuencia de chipping que se transmiten en paralelo a través del rango de frecuencia. Se consiguen velocidades de transmisión de 1Mbps a 2Mbps en la versión normal, y hasta 11 Mbps en la versión HR/DSSS. Opera en la banda de los 2,4GHz.
- OFDM (*Orthogonal Frequency Division Multiplexing*, multiplexación por división de frecuencia ortogonal): Divide una portadora de datos de alta velocidad en varias subportadoras de más baja velocidad, que luego se transmiten en paralelo. OFDM utiliza el espectro de manera mucho más eficiente, espaciando los canales a una distancia mucho menor. El espectro es más eficiente porque todas las portadoras son ortogonales entre sí, evitando de esa forma la interferencia entre portadoras muy cercanas. Se consiguen velocidades de transmisión de hasta 54Mbps. Opera en la banda de los 5GHz.

Fecha	Estándar	Banda(GHz)	Ancho de banda (MHz)	Modulación	Tasa binaria máxima
1997	802.11	2.4	20	DSSS, FHSS	2Mbps
1999	802.11b	2.4	20	DSSS	11Mbps
1999	802.11a	5	20	OFDM	54Mbps
2003	802.11g	2.4	20	DSSS, OFDM	54Mbps
2009	802.11n	2.4, 5	20, 40	OFDM	600Mbps

Tabla 2.2 Estándares 802.11.

La capa de enlace de la especificación 802.11 está compuesta por dos subcapas:

- LLC: Capa que se ocupa del control del enlace lógico. Se trata de la capa subcapa superior. Define cómo pueden acceder múltiples usuarios a la capa MAC, pues provee mecanismos de multiplexado que hacen posible que diferentes protocolos (como IP, IPX, Decnet...) puedan coexistir dentro de una red multipunto y se transporten sobre el mismo medio de red. También provee mecanismos de manejo de errores ARQ (*Automatic Repeat Request*, Repetición automática de petición).
- MAC: Conjunto de protocolos que controlan cómo los distintos dispositivos comparten el uso del espectro radioeléctrico. Es más compleja que las de otras especificaciones (802.3, 802.5, etc.). En WLAN se utiliza CSMA/CA (acceso múltiple con detección de portadora y colisión evitable). Sus funciones principales son la exploración (envío de *beacons*, balizas, que incluyen los SSID), la autenticación, asociación (el proceso que da acceso a la red), la seguridad y la fragmentación (la capacidad del punto de acceso de dividir la información en tramas más pequeñas).

2.2.1 Medida de la potencia de la señal WiFi

El denominado por sus siglas en inglés RSSI (*Received Signal Strength Indicator*, Indicador de la fuerza de la señal recibida) es una medida de la potencia presente en una señal radio en un receptor. Se suele medir a frecuencia intermedia (FI), antes del amplificador FI. En sistemas sin FI, se mide antes del amplificador en banda base.

En los sistemas IEEE 802.11, el RSSI es la potencia de señal recibida en un entorno inalámbrico, en unidades arbitrarias. Es un indicador del nivel de potencia recibido por el sistema radio tras las pérdidas de la antena, conectores y cable. Por lo tanto, mientras mayor sea el valor del RSSI, con mayor potencia llegará la señal. Los valores del RSSI se presentan como números negativos, siendo 0 el máximo y representando un 100 % de señal útil recibida. Este valor es calculado normalmente por las tarjetas de red, con frecuencia para determinar si la potencia de la señal está por encima de cierto umbral para poder comenzar la comunicación. No existe, sin embargo, ningún tipo de relación estandarizada entre los parámetros físicos y una lectura del RSSI. El estándar 802.11 no define ninguna relación entre el valor del RSSI y el nivel de potencia en milivatios o decibelios, pero sí que, dentro de los mismos, hay que tener en cuenta que a diferencia de otros sistemas de acceso por multiplexión en el tiempo (*Time Division Multiplexing Access*, TDMA), como GSM, donde para realizar este tipo de medidas de potencia se ajusta a la máscara temportal correspondiente, en 802.11 la medida se realiza durante el preámbulo de la trama, y no durante toda la ráfaga.

En el caso del sistema Android que nos ocupa en este proyecto, el propio sistema operativo cuenta con drivers para comunicarse con el chip BCM4339 de Broadcom (el que utiliza el LG G3), que contiene las

antenas WiFi y Bluetooth del teléfono, cuyo firmware se encarga de realizar la medida del RSSI. La API de Android se encarga de proporcionar a los desarrolladores un objeto `WifiManager` a través del cual obtener información sobre las redes inalámbricas detectadas por la antena, entre las cuales se encuentra la potencia recibida en decibelios. Comparando los distintos valores para las distintas redes, el dispositivo se conectará a la red cuya señal recibida tenga mayor potencia, como se detallará más adelante en el capítulo 4.

2.3 Modos Ad hoc e infraestructura

Los elementos que intervienen en cualquier comunicación inalámbrica son los puntos de acceso (equipos que dan acceso a la red de forma centralizada) y los clientes (dispositivos inalámbricos). Existen dos tipos de topologías:

- Los clientes se conectan directamente entre ellos para comunicarse (modo *Ad Hoc*).
- Los clientes se conectan a un punto de acceso para comunicarse con el exterior o entre ellos mismos (modo Infraestructura).

2.3.1 Modo Ad Hoc

En el modo Ad Hoc los equipos inalámbricos se conectan entre sí para formar una red punto a punto, es decir, los clientes intercambian la información entre ellos directamente sin pasar por ningún equipo intermedio. Las opciones de configuración de seguridad, nombre de red y canal de comunicación se configuran el propio cliente. Una vez que los clientes pertenecen a la misma red, se transmiten los datos al aire y los otros dispositivos reciben y reenvían la información. La configuración que forman los clientes se llama conjunto de servicio básico independiente o IBSS (Independent Basic Service Set).

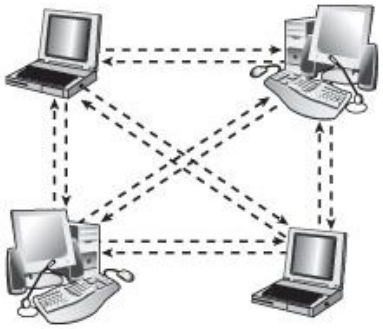

Figura 2.2 Modo ad hoc.

2.3.2 Modo Infraestructura

En el modo de infraestructura, los clientes deben comunicarse con el punto de acceso para poder utilizar la red, ya que el punto de acceso es el encargado de gestionar la autorización. Al grupo formado por el punto de acceso y los clientes que se encuentran dentro de la misma zona de cobertura se le llama conjunto de servicio básico o BSS (*Basic Service Set*). También puede darse el caso de que una red esté formada por varios puntos de acceso y clientes que se conecten a ellos. A este grupo de puntos de acceso y clientes se le llama conjunto de servicio extendido o ESS (Extended Service Set). En este tipo de redes el nombre de la red se define en el punto de acceso con el parámetro ESSID (Extended Service Set ID), lo nos permite diferenciar una red de otra.

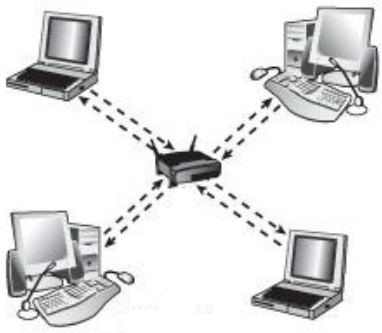

Figura 2.3 Modo infraestructura.

2.4 GPS

El sistema de posicionamiento global, llamado GPS (*Global Positioning System*), es un sistema de navegación espacial por satélite que permite determinar una posición en la Tierra con gran precisión. Desarrollado, instalado y empleado por el Departamento de Defensa de los Estados Unidos, el sistema GPS se sirve de 24 satélites y utiliza la trilateración: para determinar la posición, el receptor deberá localizar al menos tres de los satélites de la red GPS, los cuales enviarán un identificador y una señal de reloj. En base a estas señales, el dispositivo sincronizará el reloj del GPS y calculará el tiempo que han tardado en llegar las señales al equipo, y de tal modo mide la distancia al satélite mediante el método de trilateración inversa, el cual se basa en determinar la distancia de cada satélite al punto de medición. Conocidas las distancias, se determina fácilmente la propia posición relativa respecto a los satélites. Conociendo además las coordenadas o posición de cada uno de ellos por la señal que emiten, se obtiene la posición absoluta o coordenadas reales del punto de medición.

2.4.1 Fórmula del haversine

La fórmula del haversine, o fórmula del semiverseno, es una ecuación que determina la distancia de círculo máximo entre dos puntos en una esfera dadas su latitud y su longitud. Se trata de un caso particular de la denominada ley de los semiversenos, que relaciona los lados y ángulos de los triángulos esféricos. La fórmula se basa en la llamada *función haversine*:

$$hav = \sin^2(\frac{\Delta\phi}{2}) + \cos\phi_1 * \cos\phi_2 * \sin^2(\frac{\Delta\lambda}{2}) \tag{2.1}$$

$$c = 2 * atan2(\sqrt{hav}, \sqrt{(1-hav)}); \tag{2.2}$$

Siendo ϕ y λ latitud y longitud respectivamente, en radianes. La distancia en metros entre ambos puntos será:

$$d = R * c \tag{2.3}$$

donde R es el radio de la tierra en metros, $R = 6371 * 10^3$.

2.5 *Test Driven Development*

Se ha decidido usar el enfoque **TDD** (*Test-Driven Development*, Desarrollo guiado por pruebas) con el fin de desarrollar la aplicación del servidor web, una técnica de diseño e implementación de software que se centra en tres pilares fundamentales [7]:

- La implementación de las funciones justas que el cliente necesita y no más.
- La minimización del número de *bugs* (errores) que llegan al software en fase de producción.
- La producción de software modular, altamente reutilizable y preparado para el cambio.

Al contrario que en otras metodologías, donde lo primero es definir la arquitectura, las distintas clases y objetos, patrones, etc, en TDD se deja que la propia implementación de pequeños ejemplos, en constantes iteraciones, haga emerger la arquitectura que es necesario utilizar. Existen numerosas características que hacen que TDD sea una herramienta importante a la hora de diseñar software: la calidad del software aumenta, el código aumenta su reutilización, y el hecho de que escribir el ejemplo (test) antes que el código nos obliga a escribir el mínimo de funcionalidad necesaria, evitando sobrediseñar.

Para llevar a cabo TDD, en primer lugar se debe definir una lista de requisitos (objetivos) y después se ejecuta el siguiente ciclo:

1. Elegir un requisito: Se elige de una lista el requisito que se cree que nos dará mayor conocimiento del problema y que a la vez sea fácilmente implementable.
2. Escribir una prueba (test): Se comienza escribiendo una prueba para el requisito. Para ello el programador debe entender claramente las especificaciones y los requisitos de la funcionalidad que está por implementar. Este paso fuerza al programador a tomar la perspectiva de un cliente considerando el código a través de sus interfaces.
3. Verificar que la prueba falla: Si la prueba no falla es porque el requisito ya estaba implementado o porque la prueba es errónea.
4. Escribir la implementación: Escribir el código más sencillo que haga que la prueba funcione.
5. Eliminación de duplicación: El paso final es la refactorización, que se utilizará principalmente para eliminar código duplicado. Se hace un pequeño cambio cada vez y luego se corren las pruebas hasta que funcionen.
6. Actualización de la lista de requisitos: Se actualiza la lista de requisitos tachando el requisito implementado. Asimismo se agregan requisitos que se hayan visto como necesarios durante este ciclo y se agregan requisitos de diseño.

Se expone a continuación un sencillo ejemplo de cómo se lleva a cabo este tipo de desarrollo:

Supongamos que se desea implementar una sencilla calculadora que, haciendo uso de un método `add`, sume dos números proporcionados en una cadena de caracteres, separados por comas. El método deberá responder a la forma `Int add(String numbers)` (en este ejemplo se usará la sintaxis del lenguaje Java, pero la metodología es aplicable a cualquier lenguaje de programación). Definimos en primer lugar las especificaciones:

- El método aceptará como parámetros 0, 1 o 2 números.
- El método devolverá 0 si se pasa una cadena vacía como parámetro.
- El método devolverá la suma de los parámetros cuando estos sean válidos.
- El método también deberá aceptar caracteres de retorno de carro como separados válidos, además de las comas.

En este caso, se ha querido limitar el número de requerimientos con el fin de mantener la simplicidad en el ejemplo. A continuación, se elige el primer requisito de la lista, y se escriben los tests:

```
import org.junit.Test;
import com.roberto.example.tdd.StringCalculator;

public class StringCalculatorTest {
    @Test(expected = RuntimeException.class)
    public final void whenMoreThan2NumbersAreUsedThenExceptionIsThrown() {
        StringCalculator.add("1,2,3");
    }
    @Test
    public final void when2NumbersAreUsedThenNoExceptionIsThrown() {
        StringCalculator.add("1,2");
        Assert.assertTrue(true);
    }
    @Test(expected = RuntimeException.class)
    public final void whenNonNumberIsUsedThenExceptionIsThrown() {
        StringCalculator.add("1,X");
    }
}
```

Se está usando en este caso la librería Junit de Java como *framework* para escribir los tests. Se han escrito tres pruebas distintas para comprobar que se cumple la especificación: en la primera, se espera que se lance una excepción del tipo `RuntimeException` cuando se proveen más de dos números como argumentos; en la segunda, se espera que cuando se proveen dos números como argumento, no se lanza ninguna excepción; y en la tercera se comprueba que cuando se provee un argumento que no es un número, se lanza una excepción del tipo `RuntimeException`.

Si intentamos ejecutar estos tests, veremos que en el caso de Java no podemos, pues la clase a la que hacen referencia las pruebas (`StringCalculator`) aún no está implementada. Implementamos entonces la misma para que cumpla los objetivos de las pruebas, de la forma más sencilla posible:

```
public class StringCalculator {
    public static final void add(final String numbers) {
        String[] numbersArray = numbers.split(",");
        if (numbersArray.length > 2) {
            throw new RuntimeException("Up to 2 numbers separated by comma (,) are allowed");
        } else {
            for (String number: numbersArray) {
                Integer.parseInt(number); // If it is not a number, parseInt will throw an exception
            }
        }
    }
}
```

Si ejecutamos los tests, veremos que ahora todos pasarán de forma correcta. Esto nos indica que hemos implementado con éxito el primero de los requisitos. Procedemos entonces al siguiente elemento de la lista de especificaciones: en este caso, que el método devuelve 0 para el caso en el que el argumento es una cadena vacía. Añadimos un nuevo test a los anteriores para cubrir este comportamiento:

```
@Test
public final void whenEmptyStringIsUsedThenReturnValueIs0() {
    Assert.assertEquals(0, StringCalculator.add(""));
}
```

`assertEquals` comprueba la igualdad entre los dos argumentos que se le pasan (en este caso comprueba que el resultado devuelto por la llamada al método con la cadena vacía como parámetros). Si ejecutamos de nuevo el conjunto de pruebas, comprobaremos que las tres anteriores siguen pasando, mientras que la nueva fallará, pues aún no está implementada. Realizamos entonces las modificaciones necesarias en el código:

```
public static final int add(final String numbers) { // Changed void to int
    String[] numbersArray = numbers.split(",");
    if (numbersArray.length > 2) {
        throw new RuntimeException("Up to 2 numbers separated by comma (,) are allowed");
    } else {
        for (String number: numbersArray) {
            if (!number.isEmpty()) {
```

```
                Integer . parseInt (number);
            }
        }
    }
    return 0; // Added return
}
```

Ejecutando las pruebas, ahora las cuatro pasarán correctamente. Podemos comprobar por tanto que no hemos alterado el comportamiento que se había implementado hasta ahora, y que también se ha implementado de forma correcta la nueva especificación. Pasamos entonces a la última especificació en la lista: que el método devuelva la suma de los números. De nuevo escribimos los casos de prueba para este comportamiento, añadiéndolos a los ya existentes:

```
@Test
public final void whenOneNumberIsUsedThenReturnValueIsThatSameNumber() {
    Assert . assertEquals (3, StringCalculator .add("3"));
}

@Test
public final void whenTwoNumbersAreUsedThenReturnValueIsTheirSum() {
    Assert . assertEquals (3+6, StringCalculator .add("3,6"));
}
```

Se quiere comprobar que la función devuelve la suma tanto cuando se pasa un sólo número como cuando se pasan dos, de nuevo haciendo uso del método `assertEquals` para comprobar la igualdad. Siguiendo el ciclo TDD, ejecutamos los tests y observamos cómo los nuevos fallan. Pasamos entonces a escribir la implementación, volviendo a cambiar el código:

```
public static int add( final String numbers) {
    int returnValue = 0;
    String [] numbersArray = numbers. split (",");
    if (numbersArray.length > 2) {
        throw new RuntimeException("Up to 2 numbers separated by comma (,) are allowed");
    }
    for ( String number : numbersArray) {
        if (!number.trim() .isEmpty()) { // After refactoring
            returnValue += Integer . parseInt (number);
        }
    }
    return returnValue ;
```

Ejecutando de nuevo las pruebas podemos comprobar que pasan correctamente, concluyendo por tanto este ejemplo. Se puede observar cómo gracias a TDD y a varias iteraciones, se ha ido escribiendo tan sólo el código que es necesario utilizar, ayudando también al mantenimiento del mismo. Si bien en este ejemplo es difícil apreciarlo, en grandes proyectos *software* no es extraño el poder cambiar una funcionalidad sin que el desarrollador se percate de su error, y es gracias a las pruebas que este tipo de errores no se pasarán por alto.

2.6 Comunicación entre cliente y servidor

La comunicación entre el cliente y el servidor se realizará sobre el protocolo HTTP (*HyperText Transfer Protocol*, Protocolo de transferencia de hipertexto). La estandarización de HTTP corre a cargo de la IETF (*Internet Engineering Task Force*) y el W3C (*Wide Web Consortium*), mediante la publicación de RFC's (*Requests For Comments*), siendo la primera publicada la número 2068 en 1997 estandarizando la primera versión del protocolo, HTTP/1.1.

HTTP es un protocolo de petición-respuesta. El cliente realiza una petición HTTP al servidor, el cual provee recursos (como ficheros HTML, por ejemplo, en la navegación web, entre otros muchos tipos) o realiza operaciones para luego devolver un mensaje de respuesta al cliente. Dicha respuesta contendrá información sobre la petición y el contenido solicitado contenido en el cuerpo (*body*). Se trata de un protocolo de nivel de aplicación, y su definición asume una capa subyacente fiable al nivel de la capa de transporte, siendo normalmente TCP (*Transmission Control Protocol*) el más utilizado. Los recursos HTTP se identifican y localizan en la red gracias a URLs (*Uniform Resource Locators*, localizadores uniformes de recursos).

El protocolo define varios métodos (a veces llamados verbos) para indentificar la acción que se quiere realizar sobre el recurso identificado. La especificación HTTP/1.1 se definen diferentes métodos: GET, HEAD, POST, PUT, DELETE, TRACE, OPTIONS, CONNECT y PATCH. En el caso de la API que se desarrollará en el presente proyecto, se ofrecen dos rutas diferentes (URLs) que habrán de ser llamadas con el método GET, y se ilustra en la **figura 2.4** un esquema de la comunicación entre el cliente Android y el servidor Javascript:

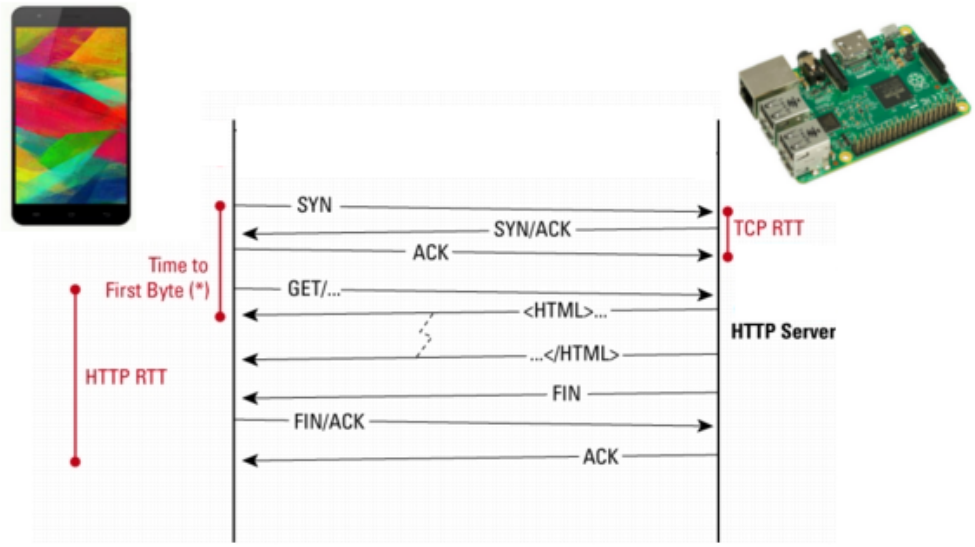

Figura 2.4 Comunicación cliente-servidor.

Se ha dejado la comunicación genérica en la figura, suponiendo respuestas HTML para indicar las diferentes posibles respuestas del servidor (que se detallarán en el capítulo 4 cuando se desarrolle el servidor web).