14 de Abril del 2012

Escáner de puertos

A. Mateos Checa, J. Ruiz Oliva, E.M. Pérez Sánchez

Módulo 11: Seguridad Informática: Estudio del caso de las Comunicaciones Móviles

MASTERINFTEL IX Edición
UNIVERSIDAD DE MÁLAGA
Patrocinador
Fundación Vodafone España
Master Universitario en Informática Aplicada a las Telecomunicaciones Móviles

Índice

- Objetivo
- Datos de red
- Localización de hosts
- Puertos TCP
- Mejoras
- Conclusiones

Índice

- **Objetivo**
- Datos de red
- Localización de hosts
- Puertos TCP
- Mejoras
- Conclusiones

Objetivo

Aplicación móvil Android que:

- Obtiene los datos de la red
- Obtiene los host de la red y sus datos
- Comprueba los puertos que tiene abiertos cada host

Además hay que usar hebras para que sea más eficiente el escaneo de los puertos.

Índice

- Objetivo
- **Datos de red**
- Localización de hosts
- Puertos TCP
- Mejoras
- Conclusiones

Datos de red

El escaneo se realiza sólo mediante conexión via Wifi.

Es posible obtener los datos de red y del dispositivo mediante el uso de las clases:

- *WifiManager*
- *WifiInfo*

Datos de red

- En resumen, los datos obtenidos de la red son:
  - IP
  - Dirección MAC
  - Velocidad del enlace (mbps)
  - RSSI (intensidad de señal en dBm)
  - SSID (nombre de la red)
  - BSSID (dirección MAC del router)
  - IP de router
  - Mascará de red

Índice

- Objetivo
- Datos de red
- **Localización de hosts**
- Puertos TCP
- Mejoras
- Conclusiones

Localización de hosts

- La búsqueda de hosts se hace mediante hebras (*AsyncTask*) de forma que se utiliza una hebra por host

- Se comprueba si el host es alcanzable con el método *isReachable* de la clase *InetAddress*. Este método hace un ping al host, es decir, manda una trama ICMP del tipo ECHO REQUEST.

- Si pertenece a la red es almacenado el host y se comprueba si es el router de la red.

Localización de hosts

- Con el método *getHostName* obtenemos el nombre.
- Para obtener la MAC de los hosts se hace uso de la tabla ARP de Linux (/proc/net/arp) que también está disponible en Android
- Los 6 dígitos primeros de la dirección Mac indican el fabricante. Estos son comparados con un archivo donde aparecen todos los fabricantes, para obtener el nombre del fabricante. Archivo obtenido de la web de la IEEE.
- A continuación se comprueban los puertos abiertos de cada uno.

Índice

- Objetivo
- Datos de red
- Localización de hosts
- **Puertos TCP**
- Mejoras
- Conclusiones

Puertos TCP

Idea:

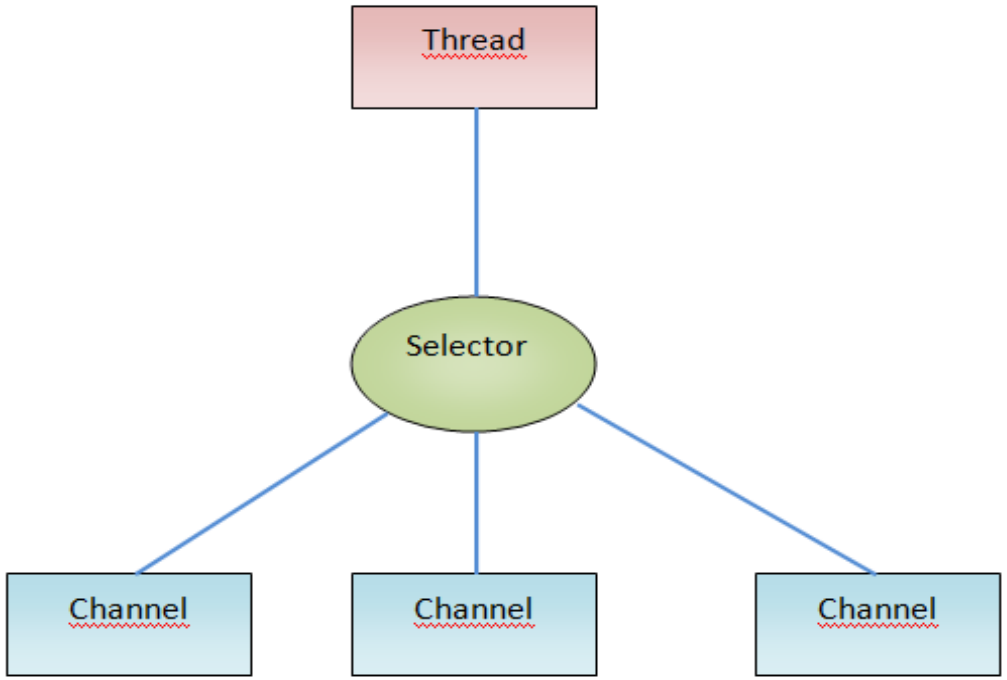
- No bloquear la aplicación durante el escaneo de puertos.
- Uso de hebras (*AsyncTask*) y el paquete *java.nio*.
- Configurable: rango de puertos, timeout...
- Identificación del nombre del puerto mediante la lista proporcionada por IANA
  www.iana.org/assignments/port-numbers

Puertos TCP

- Paquete *java.nio* permite operaciones de E/S asíncronas sin bloqueo.
- Mediante selectores se puede examinar uno o más canales y determinar qué canales están listos para leer o escribir.
- Un solo hilo puede gestionar múltiples canales, y por lo tanto varias conexiones de red.

Puertos TCP

Ilustración de un selector manejando 3 canales



Puertos TCP

- La clase encargada de realizar el escáner heredará de *AsyncTask*.
- *La clase SocketChannel* se encargará de realizar las conexiones TCP

```
SocketChannel socket = SocketChannel.open();
socket.configureBlocking(false);
socket.connect(new InetSocketAddress(ina, port));
```

- Los canales serán registrados con un selector, que nos permitirá manejar las múltiples conexiones.

```
socket.register(selector,selectionKey.OP_CONNECT, data);
```

Puertos TCP

- Además podemos indicar el tipo de evento que queremos escuchar.

```
SelectionKey.OP_CONNECT
SelectionKey.OP_ACCEPT
SelectionKey.OP_READ
SelectionKey.OP_WRITE
```

- El rango de puertos ha escanear puede ser relativamente amplio. Por lo que hay que tener cuidado de no pasar de 127 hebras.
- Una vez que el selector está abierto y con los canales registrados, llamamos al metodo *select(TIME_OUT)* que bloquea hasta que el evento registrado o el time_out son disparados.

```
if (selector.select(TIMEOUT_SELECT) > 0)
```

Puertos TCP

- El método *select* devuelve el número de canales que están listos.
- Si el número es mayor que 0, solo queda discriminar cuales son los canales que se encuentran abiertos.

```
Iterator<SelectionKey> iterator = selector.selectedKeys().iterator();
 while (iterator.hasNext()) {
        SelectionKey key = (SelectionKey) iterator.next();
       if (key.isConnectable()) {
              if (((SocketChannel) key.channel()).finishConnect()) {
                    finishKey(key, OPEN);
       ...
```

- Cuando un puerto está abierto, se notifica.

```
publishProgress(data.port, state, banner);
```

- En caso contrario, el timeout ha sido disparado.

Índice

- Objetivo
- Datos de red
- Localización de hosts
- Puertos TCP
- **Mejoras**
- Conclusiones

Mejoras

- Escanear puertos UDP: enviar datagrama al puerto indicado y esperar la contestación durante un timeout.
- Indicar el rango de host a buscar
- Identificar el SO de los hosts.
- Uso de la librería *libpcap*, que permite al usuario controlar paquetes a bajo nivel.
- Mejorar la velocidad del parseo de los ficheros de texto de fabricantes y puertos

Índice

- Objetivo
- Datos de red
- Localización de hosts
- Puertos TCP
- Mejoras
- **Conclusiones**

Conclusiones

- Java es un lenguaje de alto nivel por lo que no es posible crear paquetes propios y raw sockets
- No todas las librerías Java son compatibles con Android
- El uso de hebras mejora el rendimiento de la aplicación considerablemente
- Ciclo de ejecución de AsyncTask es bastante intuitivo y práctico comparada con el uso de la clase Handler.

FIN

¡Gracias por su atención!

MASTERINFTEL
IX Edición
UNIVERSIDAD DE MÁLAGA
Patrocinador
Fundación Vodafone España
Master Universitario en Informática Aplicada a las Telecomunicaciones Móviles