Android port forwarding per la comunicazione tra due emulatori

Scritto da Francesco Abate

Basato su una intensa ricerca nel web di ben 2 mesi

GitHub: https://github.com/frekkanzer2

LinkedIn: https://www.linkedin.com/in/francescoabateimtech/

Tale guida non mostra come realizzare una app che comunichi con un'altra app, bensì tende a fornire una soluzione per far comunicare due emulatori tra loro.

Supponiamo di possedere un'applicazione che crea un server ed un'altra che fa da client, quindi che cerca di connettersi al server. Prendiamo un'applicazione di testing da me creata e disponibile pubblicamente sulla mia repository di GitHub dedicata ad Android.

Il server viene creato in tal modo:

```
try{
    myServer = new ServerSocket(SERVERPORT);
}catch(IOException e){
    System.err.println("*********************************************");
    e.printStackTrace();
    System.err.println("Problem creating ServerSocket");
}
```

La variabile SERVERPORT è ovviamente stabilita dal programmatore. In tal caso, assume valore 5000.

Creato il server, rimarrà in attesa di connessioni da parte di un qualsiasi client.

```
Socket clientConnection = null;
try{
    clientConnection = myServer.accept();
}catch(IOException e){
    System.err.println("*********************************************");
    e.printStackTrace();
    System.err.println("Problem accepting client connection");
}
```

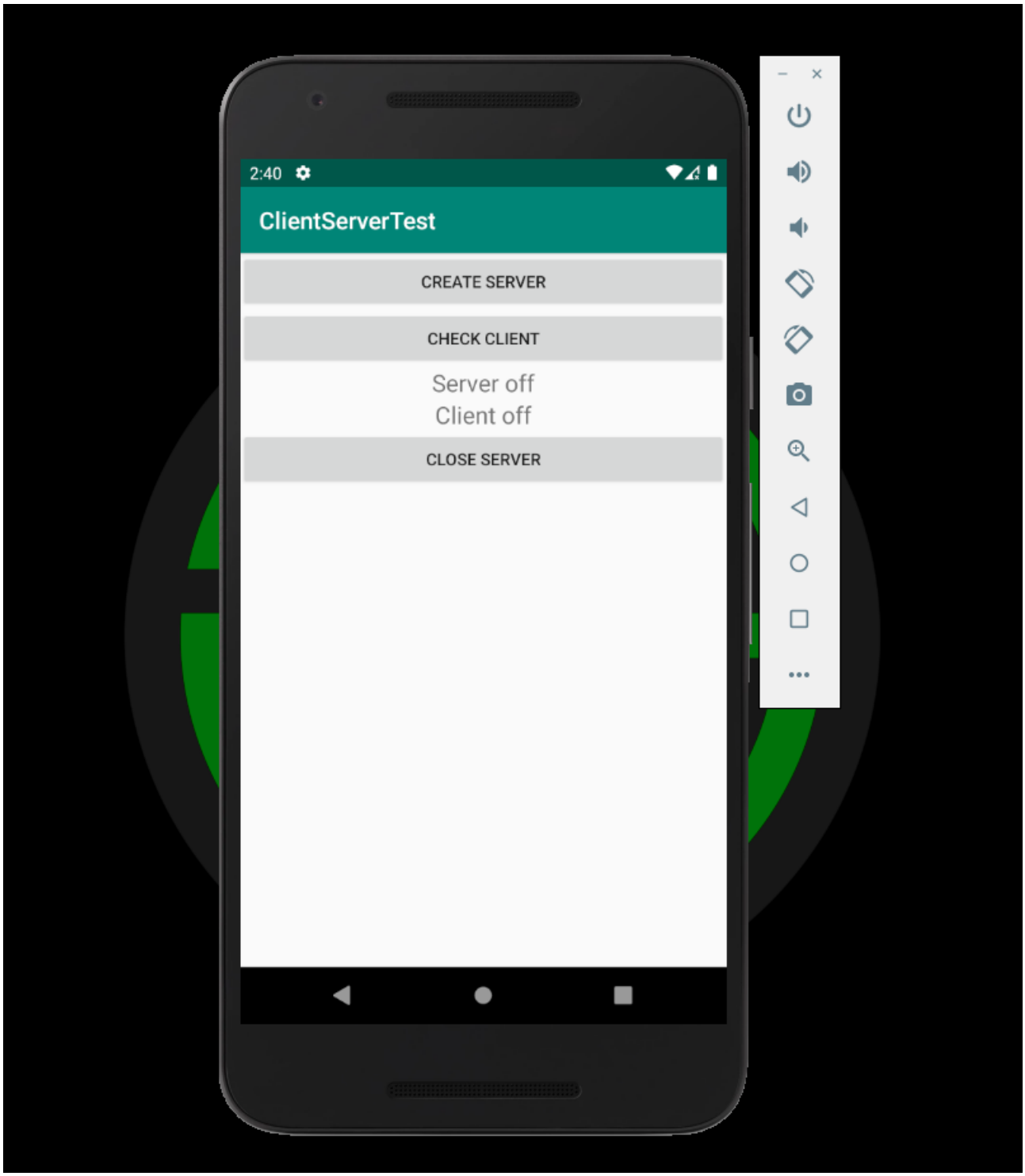
Il client, invece, cercherà di connettersi con il seguente codice:

```
try{
    System.err.println("Connecting...");
    MainActivity.changeStatusClient( msg: "Connecting to server...");
    connection = new Socket(InetAddress.getByName ("10.0.2.2"), CLIENTPORT);
    MainActivity.changeStatusClient( msg: "Client connected");
    System.err.println(connection.getInetAddress().toString());
    System.err.println(connection.getLocalPort());
    System.err.println("CONNECTED");
}catch(IOException e){
    System.err.println("*********************************************");
    System.err.println(e.getMessage());
    System.err.println("*********************************************");
    e.printStackTrace();
    System.err.println("*********************************************");
    System.err.println("ERR > Problem with connection");
}
```

La chiave fondamentale di tale codice è dove viene creato il socket: lì è dove il client proverà a connettersi al server. La variabile CLIENTPORT è ovviamente stabilita dal programmatore. In tal caso, assume valore 6000.

È importante che SERVERPORT sia diversa da CLIENTPORT e, per convenzione, CLIENTPORT > SERVERPORT.

Si inizia aprendo l'emulatore che creerà il server. Avviamo solamente l'applicazione, ma non il server (si potrebbe anche avviare in realtà, ma non è sicuro funzioni).



Avviato l'emulatore che farà da server, apriamo il terminale del sistema operativo che si sta utilizzando e controlliamo se adb è registrato nei path del sistema: per fare ciò, basta inserire il comando adb nel terminale e vedere se viene rilevato.

Prompt dei comandi

```
C:\Users\abate>adb
Android Debug Bridge version 1.0.41
Version 29.0.4-5871666
Installed as C:\Users\abate\AppData\Local\Android\Sdk\platform-tools\adb.exe

global options:
 -a         listen on all network interfaces, not just localhost
 -d         use USB device (error if multiple devices connected)
 -e         use TCP/IP device (error if multiple TCP/IP devices available)
 -s SERIAL  use device with given serial (overrides $ANDROID_SERIAL)
 -t ID      use device with given transport id
 -H         name of adb server host [default=localhost]
 -P         port of adb server [default=5037]
 -L SOCKET  listen on given socket for adb server [default=tcp:localhost:5037]

general commands:
 devices [-l]             list connected devices (-l for long output)
 help                     show this help message
 version                  show version num

networking:
 connect HOST[:PORT]      connect to a device via TCP/IP
 disconnect [[HOST]:PORT] disconnect from given TCP/IP device, or all
 forward --list           list all forward socket connections
 forward [--no-rebind] LOCAL REMOTE
     forward socket connection using:
       tcp:<port> (<local> may be "tcp:0" to pick any open port)
       localabstract:<unix domain socket name>
       localreserved:<unix domain socket name>
       localfilesystem:<unix domain socket name>
       dev:<character device name>
       jdwp:<process pid> (remote only)
 forward --remove LOCAL   remove specific forward socket connection
 forward --remove-all     remove all forward socket connections
 ppp TTY [PARAMETER...]   run PPP over USB
 reverse --list           list all reverse socket connections from device
 reverse [--no-rebind] REMOTE LOCAL
     reverse socket connection using:
       tcp:<port> (<remote> may be "tcp:0" to pick any open port)
       localabstract:<unix domain socket name>
       localreserved:<unix domain socket name>
       localfilesystem:<unix domain socket name>
 reverse --remove REMOTE  remove specific reverse socket connection
 reverse --remove-all     remove all reverse socket connections from device

file transfer:
 push [--sync] LOCAL... REMOTE
     copy local files/directories to device
     --sync: only push files that are newer on the host than the device
 pull [-a] REMOTE... LOCAL
     copy files/dirs from device
     -a: preserve file timestamp and mode
 sync [all|data|odm|oem|product|system|system_ext|vendor]
     sync a local build from $ANDROID_PRODUCT_OUT to the device (default all)
     -l: list files that would be copied, but don't copy them

shell:
 shell [-e ESCAPE] [-n] [-Tt] [-x] [COMMAND...]
     run remote shell command (interactive shell if no command given)
     -e: choose escape character, or "none"; default '~'
     -n: don't read from stdin
     -T: disable PTY allocation
```

Insomma, ci sono tante belle cose che per quel che dobbiamo fare non ci interessano: l'importante è sapere che adb sia presente nei path del sistema. In caso contrario, basta effettuare una ricerca su Google e trovare una semplice guida che mostri come inserire adb nei path del sistema (ovviamente è d'obbligo cercare in inglese).

Quel che ci serve è creare un "ponte" tra la porta del client e quella del server, permettendo la comunicazione. Per fare ciò, è necessario eseguire il seguente comando:

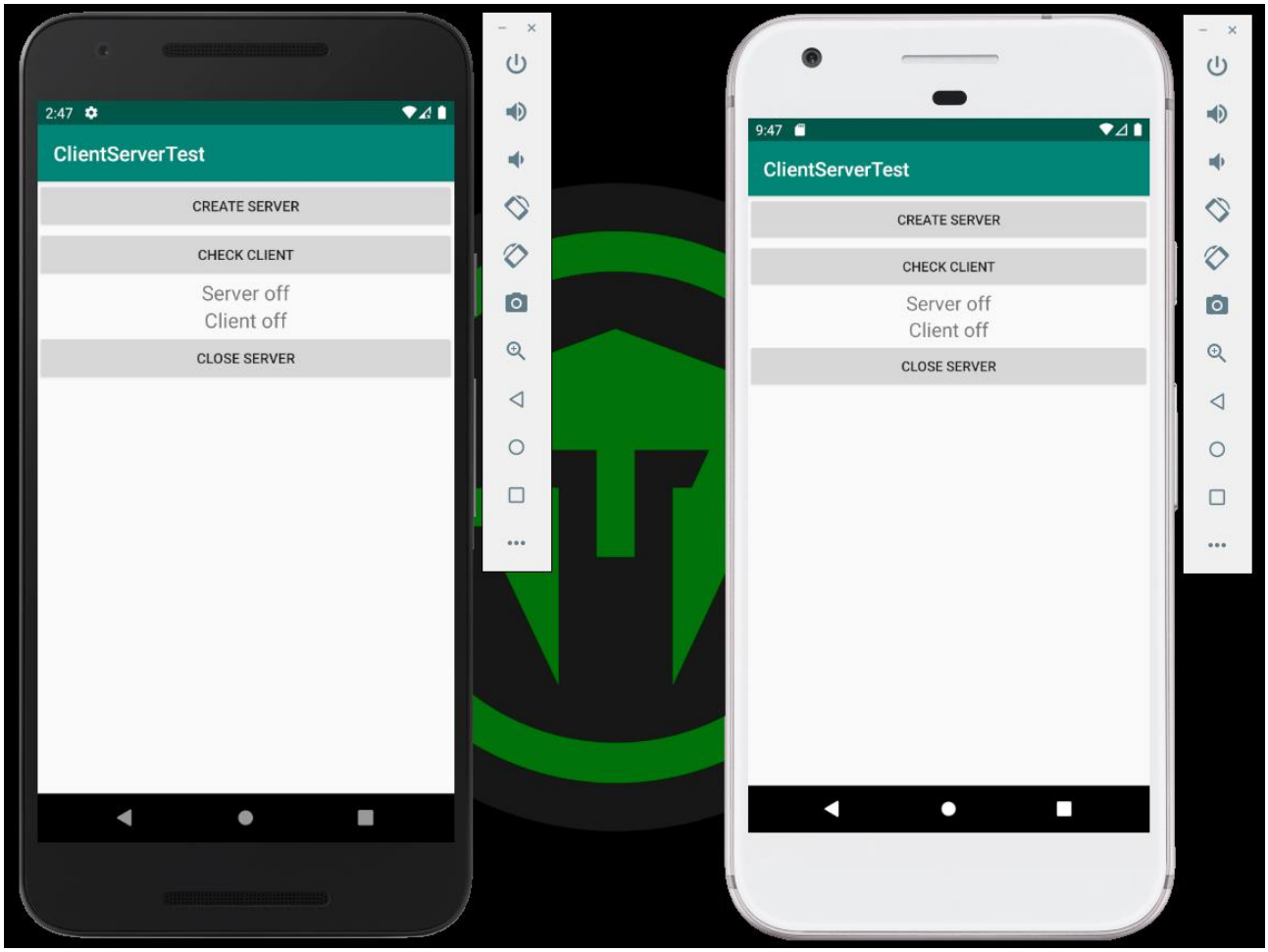
adb forward tcp:CLIENTPORT tcp:SERVERPORT

Ovviamente, ripetiamo, deve essere aperto solamente l'emulatore che ospita il server (non ancora avviato). Inoltre, CLIENTPORT e SERVERPORT dovranno assumere i valori da voi indicati. Da esempio, viene eseguito:

```
C:\Users\abate>adb forward tcp:6000 tcp:5000
6000
```

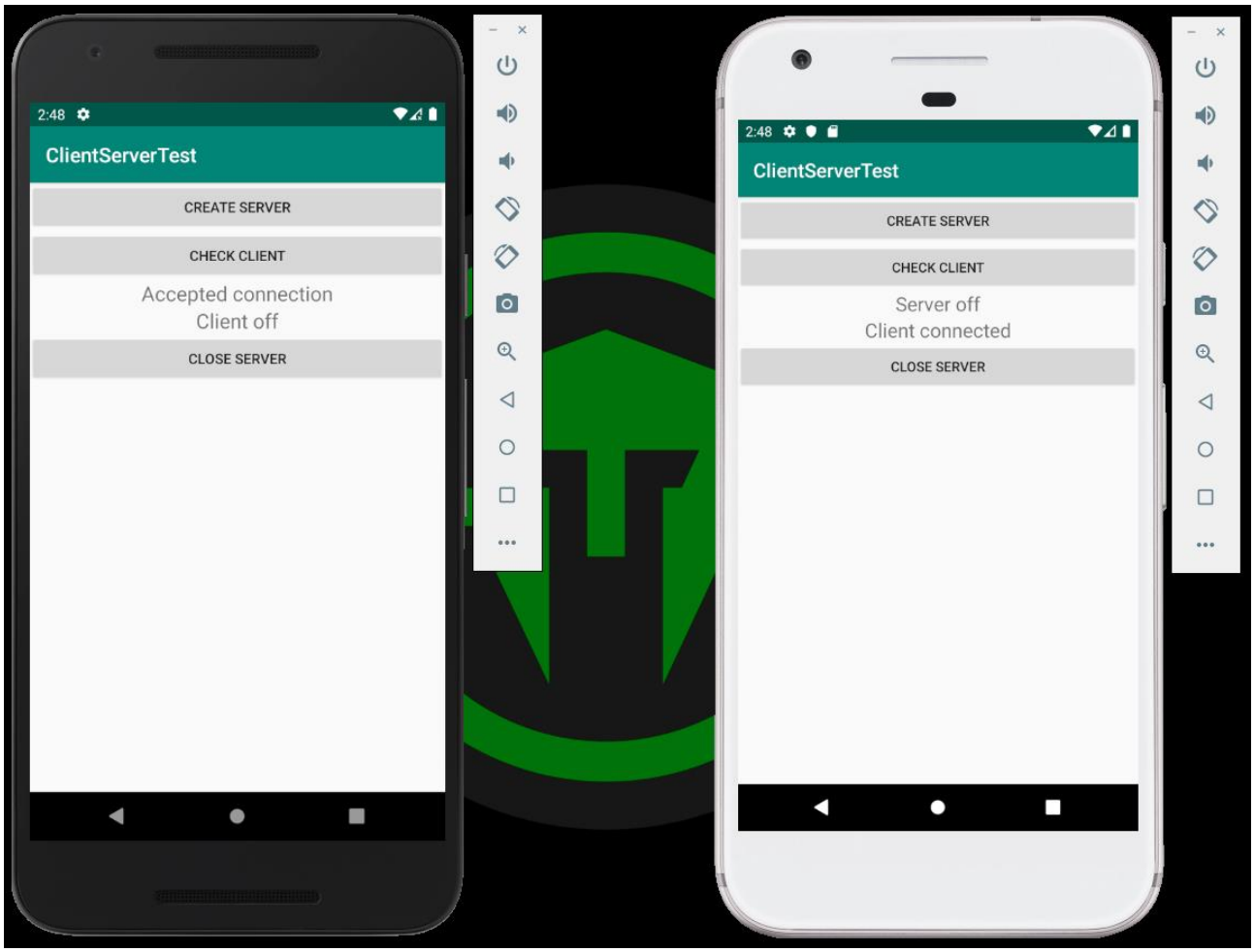
6000 nella newline è l'output

Effettuato il forward, è possibile aprire anche il secondo emulatore.



Finalmente possiamo avviare il server e controllare se il client si connette!

Ovviamente il risultato sarà positivo.



Alcune guide suggeriscono l'utilizzo di Telnet: personalmente l'ho provato e non ha funzionato, nonostante dovesse effettuare la stessa identica funzione di adb. Inoltre, Telnet richiede ad ogni riavvio dell'emulatore una password situata in un file, mentre adb ne fa a meno. In tal caso, adb si dimostra decisamente migliore e più semplice. Quindi, almeno per questa operazione, sconsiglio vivamente Telnet.

Sia chiaro che non sto dicendo che Telnet sia inutile o non conveniente: semplicemente, per tale operazione, risulta esser molto più semplice l'utilizzo di adb.

Francesco Abate

https://www.linkedin.com/in/francescoabateimtech/

https://github.com/frekkanzer2