Android Network Programming: Introduction

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science
Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Introduction

• Explore the motivations for & challenges of networked software

*Networked software* defines protocols that enable computing devices to exchange messages & perform services remotely.
Introduction

• Explore the motivations for & challenges of networked software

• Describe the Android mechanisms available to implement apps & services that communicate across process boundaries that span mobile devices & server hosts
Introduction

• Explore the motivations for & challenges of networked software
• Describe the Android mechanisms available to implement apps & services that communicate across process boundaries
• Many Android apps use Sockets & TCP and/or HTTP to communicate & exchange data via the Internet
  • e.g., Browser, Email, MMS/SMS, Calendar, Contacts, etc.

See developer.android.com/training/basics/network-ops/connecting.html
Introduction

• Explore the motivations for & challenges of networked software
• Describe the Android mechanisms available to implement apps & services that communicate across process boundaries
• Many Android apps use Sockets & TCP and/or HTTP to communicate & exchange data via the Internet
• Android also provides certain IPC mechanisms that are optimized for inter-process communicate within a mobile device
  • e.g., the Android Interface Definition Language (AIDL) & Binder framework

We’ll cover the the Android Binder & AIDL in later modules
Learning Objectives in this Part of the Module

- Understand the motivations for & challenges of networked software
Motivations for Networked Software

• Collaboration & commerce
  • e.g., file sharing, social media, e-commerce online transaction processing, B2B supply chain management, etc.
Motivations for Networked Software

- Collaboration & commerce
- Scalability
  - e.g., utility computing in clouds

www.ge.com/stories/industrial-internet has many apt examples
Motivations for Networked Software

- Collaboration & commerce
- Scalability
- Availability
  - e.g., minimizing single points of failure via replication
Motivations for Networked Software

- Collaboration & commerce
- Scalability
- Availability
- Cost effectiveness
  - e.g., via shared resources
Challenges for Networked Software

- **Accidental Complexities**
- **Algorithmic decomposition**

Algorithmic decomposition is a historically popular design method that structures the software based on the actions performed by the system.

Challenges for Networked Software

- Accidental Complexities
  - Algorithmic decomposition
  - Continuous re-discovery & re-invention of core components

See [steve.vinoski.net/pdf/IEEE-Middleware_Dark_Matter.pdf](http://steve.vinoski.net/pdf/IEEE-Middleware_Dark_Matter.pdf) for more
Challenges for Networked Software

- **Accidental Complexities**
  - Algorithmic decomposition
  - Continuous re-discovery & re-invention of core components

- **Inherent Complexities**
  - Latency & jitter
Challenges for Networked Software

- Accidental Complexities
  - Algorithmic decomposition
  - Continuous re-discovery & re-invention of core components

- Inherent Complexities
  - Latency & jitter
  - Reliability & partial failure

Error detection & handling is more complicated for networked software
Challenges for Networked Software

- **Accidental Complexities**
  - Algorithmic decomposition
  - Continuous re-discovery & re-invention of core components

- **Inherent Complexities**
  - Latency & jitter
  - Reliability & partial failure
  - Security

See [www.dre.vanderbilt.edu/~schmidt/PDF/pdcp-editorial.pdf](http://www.dre.vanderbilt.edu/~schmidt/PDF/pdcp-editorial.pdf) for more info
Summary

- Networked software helps
- Leverage advances in hardware & networking technology
- Meet the quality & performance needs of apps & services
Summary

• Networked software helps
• Leverage advances in hardware & networking technology
• Meet the quality & performance needs of apps & services

• Successful networked software solutions must address key *accidental* & *inherent* complexities arising from
  • Limitations with development tools/techniques
  • Fundamental domain challenges
Summary

- Networked software helps
  - Leverage advances in hardware & networking technology
  - Meet the quality & performance needs of apps & services
- Successful networked software solutions must address key *accidental* & *inherent* complexities arising from
  - Limitations with development tools/techniques
  - Fundamental domain challenges
- As networked systems have grown in scale & functionality they must cope with a broader & more challenging set of complexities
Android Network Programming: Part 2

Douglas C. Schmidt
d.schmidt@vanderbilt.edu
www.dre.vanderbilt.edu/~schmidt

Professor of Computer Science
Institute for Software Integrated Systems
Vanderbilt University
Nashville, Tennessee, USA
Learning Objectives in this Part of the Module

• Understand the foundational network programming mechanisms in Android
Overview of Android Network Programming

- Android includes multiple network programming classes, e.g.,
  - java.net – (Socket, URL, etc.)
  - org.apache – (HttpRequest, HttpResponse, etc.)
  - android.net – (AndroidHttpClient, URI, AudioStream, etc.)
Overview of Android Network Programming

- Android includes multiple network programming classes, e.g.,
  - java.net – (Socket, URL, etc.)
  - org.apache – (HttpRequest, HttpResponse, etc.)
  - android.net – (AndroidHttpClient, URI, AudioStream, etc.)
- Under the hood, Android’s HTTP libraries use the Java Sockets API
  - A socket is a software endpoint that can create a bi-directional “reliable” communication link between software processes

Sockets are a common programming interface for network communication
Overview of Android Network Programming

- Android includes multiple network programming classes, e.g.,
  - java.net – (Socket, URL, etc.)
  - org.apache – (HttpRequest, HttpResponse, etc.)
  - android.net – (AndroidHttpClient, URI, AudioStream, etc.)
- Under the hood, Android’s HTTP libraries use the Java Sockets API.
- Even deeper under the hood, Android’s java.net implementation uses the Linux C Socket API via JNI.

en.wikipedia.org/wiki/Java_Native_Interface has more info on JNI.
Overview of Java Sockets

1: Passive Role

ServerSocket
accept()
bind()
close()

Socket

Network

represents a server-side factory that waits for incoming client connections & creates connected Sockets

developer.android.com/reference/java/net/ServerSocket.html
Overview of Java Sockets

1: Passive Role

ServerSocket
- accept()
- bind()
- close()

Socket

Network

Plays a portion of the Acceptor role in the Acceptor-Connector pattern

www.dre.vanderbilt.edu/~schmidt/PDF/Acc-Con.pdf has more info
Overview of Java Sockets

Provides a client-side (& server-side) TCP socket

2: Active Role

Socket
bind()
close()
connect()
getInputStream()
getOutputStream()

1: Passive Role

ServerSocket
accept()
bind()
close()

Socket

Network

developer.android.com/reference/java/net/Socket.html
Overview of Java Sockets

Clients & servers designate their addresses with an InetAddress

2: Active Role

Socket
bind()
close()
connect()
getInputStream()
getOutputStream()

1: Passive Role

ServerSocket
accept()
bind()
close()

Network

developer.android.com/reference/java/net/InetAddress.html
Overview of Java Sockets

1: Passive Role
- ServerSocket
  - accept()
  - bind()
  - close()

2: Active Role
- Socket
  - bind()
  - close()
  - connect()
  - getInputStream()
  - getOutputStream()

3: Communication Role
- InputStream & OutputStream are used to exchange bytes on connected sockets

[Image of a tablet and a server]

oreilly.com/catalog/javaio/chapter/ch05.html has more info
InputStream in Android

- An InputStream is a stream of incoming byte data
- An InputStream can be obtained from a Socket by using the `getInputStream()` method
- To read from a stream, you must create a byte buffer to read in data
- Each call to read on an InputStream fills your buffer with data & returns the number of bytes read

```java
InputStream in = someSocket.getInputStream();
const int BUFSIZ = 1024;
byte[] buffer = new byte[BUFSIZ];

for(int bytesRead;
    (bytesRead = in.read(buffer,0,buffer.length)) != -1;
) {
    // the buffer’s been filled, do something with the data
}
```

developer.android.com/reference/java/io/InputStream.html has more info
InputStreamReader in Android

• An InputStreamReader turns a byte stream into a character stream
• Data read from the source input stream is converted into characters by either a default or a provided character converter
• InputStreamReader contains an 8K buffer of bytes read from the source stream & converts these into characters as needed

```java
InputStream in = someSocket.getInputStream();
Reader reader = new InputStreamReader(in);

for (int data; (data = reader.read()) != -1; ){
    char theChar = (char) data;
    // … do something with the data
}

reader.close();
```
InputStreamReader in Android

- An InputStreamReader turns a byte stream into a character stream.
- Data read from the source input stream is converted into characters by either a default or a provided character converter.
- InputStreamReader contains an 8K buffer of bytes read from the source stream & converts these into characters as needed.

```java
InputStream in = someSocket.getInputStream();
Reader reader = new InputStreamReader(in);

for (int data; (data = reader.read()) != -1; ){
    char theChar = (char) data;
    // ... do something with the data
}
```

---

Can also read a buffer at a time

developer.android.com/reference/java/io/InputStreamReader.html has more
BufferedReader in Android

• Wraps an existing Reader & buffers the input
• Expensive interaction with underlying reader is minimized, since most (smaller) requests can be satisfied by accessing buffer alone
• Drawback is that some extra space is required to hold the buffer & copying takes place when filling that buffer

BufferedReader bufferedReader =
    new BufferedReader(new InputStreamReader
    (someSocket.getInputStream()));

for (String data;
    (data = bufferedReader.readLine()) != null;
    ){
    // ... do something with the data
}

bufferedReader.close();

developer.android.com/reference/java/io/BufferedReader.html has more info
OutputStream in Android

- An OutputStream is a stream of outgoing byte data
- An OutputStream can be obtained from a Socket by using the getOutputStream() method
- You can write data to a stream by passing in a byte buffer of data
- You should use flush() if you want to ensure the data you have written is output to disk or sent to other end of socket

```java
OutputStream out = someSocket.getOutputStream();
out.write("Hello Socket".getBytes());
out.flush();

byte[] buffer = new byte[1024];
// fill the buffer
out.write(buffer,0,buffer.length);
out.close();
```

developer.android.com/reference/java/io/OutputStream.html has more info
OutputStreamWriter in Android

- A class for turning a character stream into a byte stream
- Data written to the target input stream is converted into bytes by either a default or a provided character converter
- OutputStreamWriter contains an 8K buffer of bytes to be written to target stream & converts these into characters as needed

```java
OutputStreamWriter out = 
    new OutputStreamWriter 
    (someSocket.getOutputStream());

String string1 = "Android socket IO",
    string2 = " is fun";

out.write(string1);
out.append(string2);
out.flush();
out.close();
```

developer.android.com/reference/java/io/OutputStreamWriter.html has more
**PrintWriter in Android**

- Wraps either an existing OutputStream or an existing Writer (including OutputStreamWriter)
- Provides convenience methods for printing common data types in a human readable format

```java
PrintWriter pw =
    new PrintWriter(new OutputStreamWriter(someSocket.getOutputStream()),
    // "true" indicates auto-flush
    true);

pw.println("GET /index.html");

BufferedReader br = new BufferedReader(new InputStreamReader(socket.getInputStream()));

for (String rawData; (rawData = br.readLine()) != null; )
    data.append(rawData);
```

[developer.android.com/reference/java/io/PrintWriter.html](https://developer.android.com/reference/java/io/PrintWriter.html) has more info
Android I/O Implements the Decorator Pattern

• The Java I/O streams classes use the *Decorator* pattern, which allows for the dynamic wrapping of objects to modify their existing responsibilities & behaviors.

![Diagram showing the Decorator pattern with classes Component, ConcreteComponent, Decorator, ConcreteDecoratorA, and ConcreteDecoratorB.]

- Stream classes extend the base subclasses to add features to the stream classes.

[en.wikipedia.org/wiki/Decorator_pattern](http://en.wikipedia.org/wiki/Decorator_pattern) has more on Decorator
public class NetworkingSocketsActivity extends Activity
{
    TextView mTextView = null;

    public void onCreate(Bundle savedInstanceState) {
        ...
        // assuming server is listening on port 80
        new HttpGet().execute("www.dre.vanderbilt.edu ");
    }

    Pass a URL to the template method of an AsyncTask
private class HttpGet
    extends AsyncTask<String, Void, String> {

    Runs in a background thread

    protected String doInBackground(String... params) {
        Socket socket = null;
        StringBuffer data = new StringBuffer();
        try {
            socket = new Socket(params[0], 80);
            PrintWriter pw =
                new PrintWriter(new
                    OutputStreamWriter(socket.getOutputStream()),
                    true);
            pw.println("GET /index.html");
            ...
            Send GET request
        } catch (IOException e) {
            // Handle exception
        }
    }
}
Programming Java Sockets in Android

```java
... BufferedReader br = new BufferedReader(
    new InputStreamReader
    (socket.getInputStream()));
String rawData;
    while ((rawData = br.readLine()) != null) {
        data.append(rawData);
    }
} catch ...
// close socket
    return data.toString();
...
}

protected void onPostExecute(String result) {
    mTextView.setText(result);
}
```

- **Read data from server**
- **Return data as a String**
- **Display the text on the screen**
Programming with URLConnection in Android

```java
public class NetworkingURLActivity extends Activity {
    TextView mTextView = null;

    public void onCreate(Bundle savedInstanceState) {
        new HttpGetTask().execute("http://api.geonames.org/...");
    }

    ...  
```

Pass a URL to the template method of an AsyncTask
private class HttpGetTask extends AsyncTask<String, Void, String> {

    protected String doInBackground(String... params) {
        StringBuffer serverData = new StringBuffer();
        HttpURLConnection conn = (HttpURLConnection) new URL(params[0]).openConnection();
        InputStream in = new BufferedInputStream(conn.getInputStream());
        BufferedReader br = ...;
        ...
        return serverData.toString();
    }

    ...}

developer.android.com/reference/java/net/URLConnection.html has more
Networking Permissions

- To allow an app to access the Internet using Eclipse, open AndroidManifest.xml, go to “Permissions” tab, add "Uses Permission" & select android.permission.INTERNET.

[developer.android.com/training/basics/network-ops/connecting.html](developer.android.com/training/basics/network-ops/connecting.html)
Networking Permissions

• To allow an app to access the Internet using Eclipse, open AndroidManifest.xml, go to “Permissions” tab, add "Uses Permission" & select android.permission.INTERNET

• Alternatively, open the AndroidManifest.xml file as raw XML & add a line near the top:

```
<manifest
    xmlns:android="http://schemas.android.com/apk/res/android"
    package="examples.threadeddownloads"
    android:versionCode="1"
    android:versionName="1.0" >

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

</manifest>
```

If you don't do this, your application will crash with an UnknownHostException when trying to access a remote host!!

ffct.cc/android-development-accessing-the-internet
Summary

- Android provides a wide range of network programming mechanisms

There are many Android tutorials & resources available online.

www.vogella.com/tutorials.html
Summary

- Android provides a wide range of network programming mechanisms
- There are many patterns underlying these Android network programming mechanisms, e.g.:
  - Wrapper Facade

www.dre.vanderbilt.edu/~schmidt/PDF/wrapper-facade.pdf has more info
Summary

- Android provides a wide range of network programming mechanisms
- There are many patterns underlying these Android network programming mechanisms, e.g.:
  - Wrapper Façade
  - Bridge

en.wikipedia.org/wiki/Bridge_pattern has more info
Summary

- Android provides a wide range of network programming mechanisms.
- There are many patterns underlying these Android network programming mechanisms, e.g.:
  - Wrapper Façade
  - Bridge
  - Decorator

Summary

- Android provides a wide range of network programming mechanisms.
- There are many patterns underlying these Android network programming mechanisms, e.g.:
  - Wrapper Façade
  - Bridge
  - Decorator
  - Acceptor-Connector

www.dre.vanderbilt.edu/~schmidt/PDF/Acc-Con.pdf has more info