CS434/534: Topics in Networked (Networking) Systems

Mobile Networking System:
Making Connections: Bluetooth; WiFi Direct; Cellular

Yang (Richard) Yang
Computer Science Department
Yale University
208A Watson
Email: yry@cs.yale.edu

http://zoo.cs.yale.edu/classes/cs434/
Admin.

- Project planning
  - **Start:** Apr. 5
    - A meeting w/ instructor; Due: team, potential topics
  - **Checkpoint I:** Apr. 13 (updated; Thursday)
    - Initial ideas; survey of related work; feedback from instructor/TF
  - **Checkpoint II:** Apr. 20 (updated)
    - Initial design; feedback from instructor/TF
  - **Checkpoint III:** Apr. 27 (updated)
    - Design refined; key components details
  - **Final report:** May 4
    - No more than 6 page write up
If one process sends data to another process, it is called transaction. The data is called transaction data.

<table>
<thead>
<tr>
<th>Target</th>
<th>Binder Driver Command</th>
<th>Cookie</th>
<th>Sender ID</th>
<th>Data:</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Target Command 0</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Target Command 1</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>...</td>
</tr>
<tr>
<td></td>
<td></td>
<td></td>
<td></td>
<td>Target Command n-1</td>
</tr>
</tbody>
</table>
Recap: Binder + Service Manager

Client

Service Manager

Server

Binder Driver: /dev/binder

Client

Service Manager

Server

User Spa

Kernel Spa

Binder Driver: /dev/binder

Client

Service Manager

Server

User Spa

Kernel Spa

Binder Driver: /dev/binder

$ adb shell service list
Recap: Network Service Discovery and Execution

Diagram:
- src -> dst: request
- sd -> dst: result
- sd -> src: location
- sd -> src: service
Recap: The Linda Paradigm

- **Naming scheme:**
  - arbitrary tuples (heterogeneous-type vectors)

- **Name resolution:**
  - Nodes write into shared memory
  - Nodes read matching tuples from shared memory
    - exact matching is required for extraction
Recap: DNS

- Clients
- DNS
  - <name, service>
  - IP address
- Routers
- Servers
Recap: DNS-Service Discovery

- **Leverage DNS message format**, but each node can announce its own services

- **Multicast in a small world**
  - no central address server
    - each node is a responder
  - link-local addressing
    - send to multicast address: 224.0.0.251

- Printer
  - IP: 169.254.10.29

- Network
  - IP: 169.254.1.219

- Smartphone
  - IP: 169.254.4.51
Outline

- Admin
- Android
  - Platform overview
  - Basic concepts
  - Inter-thread: execution model with multiple threads
  - Inter-process: component composition
  - Inter-machine: network-wise composition
    - Service discovery
    - Make connections
      - Making standard connection: TCP Socket
TCP Socket Big Picture

- Welcome socket: the waiting room
- connSocket: the operation room

Client process

Server process

Client socket

Connection socket

Three-way handshake

bytes

bytes
Client/server Socket Workflow: TCP

Server (running on hostid)

create socket, port=x, for incoming request:
welcomeSocket = ServerSocket(x)

wait for incoming connection request:
connectionSocket = welcomeSocket.accept()

read request from connectionSocket
write reply to connectionSocket
close connectionSocket

Client

create socket, connect to hostid, port=x
clientSocket = Socket()

send request using clientSocket
read reply from clientSocket
close clientSocket
Outline

- Admin
- Android
  - Platform overview
  - Basic concepts
  - Inter-thread: execution model with multiple threads
  - Inter-process: component composition
  - Inter-machine: network-wise composition
    - Service discovery
    - Make connections
      - Making standard connection: TCP Socket
      - Bluetooth
Recall: Bluetooth Physical Layer

- Nodes form piconet: one master and up to 7 slaves
  - Each radio can function as a master or a slave
- The slaves discover the master and follow the pseudorandom jumping sequence of the master

A piconet
Bluetooth Details

- Each BT device is identified by a 48 bit MAC address

- Operation
  - Device Discovery
    - Inquiry scan (Sender. at faster jumping freq than recv)
    - Connection
      - Inquiry scan: get device address; Page Scan: Use dest hop seq
  - Pairing
    - Authentication/share keys; after pairing, they become bonded
Bluetooth Profiles

Example: Handfree Profile
Bluetooth Software Framework Design in Android

- Discussion: How may Bluetooth be supported in Android?

Bluetooth Software Framework Design in Android

- Announce permission

```xml
<manifest ...
<uses-permission android:name="android.permission.BLUETOOTH" />
<uses-permission android:name="android.permission.BLUETOOTH_ADMIN" />
...
</manifest>
```

- Use intents to turn on BT, request discoverable

```java
Intent enableBtIntent = new Intent(BluetoothAdapter.ACTION_REQUEST_ENABLE);
startActivityForResult(enableBtIntent, REQUEST_ENABLE_BT);

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

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

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

```java
@Override
protected void onCreate(Bundle savedInstanceState) {
    ...

    // Register for broadcasts when a device is discovered.
    IntentFilter filter = new IntentFilter(BluetoothDevice.ACTION_FOUND);
    registerReceiver(mReceiver, filter);
}

// Create a BroadcastReceiver for ACTION_FOUND.
private final BroadcastReceiver mReceiver = new BroadcastReceiver() {
    public void onReceive(Context context, Intent intent) {
        String action = intent.getAction();
        if (BluetoothDevice.ACTION_FOUND.equals(action)) {
            // Discovery has found a device. Get the BluetoothDevice
            // object and its info from the Intent.
            BluetoothDevice device = intent.getParcelableExtra(BluetoothDevice.EXTRA_DEVICE);
            String deviceName = device.getName();
            String deviceHardwareAddress = device.getAddress(); // MAC address
        }
    }
};
```
Mutator: Connecting as Server

```java
private class AcceptThread extends Thread {
    private final BluetoothServerSocket mmServerSocket;

    public AcceptThread() {
        // Use a temporary object that is later assigned to mmServerSocket
        // because mmServerSocket is final.
        BluetoothServerSocket tmp = null;
        try {
            // MY_UUID is the app's UUID string, also used by the client code.
            tmp = mBluetoothAdapter.listenUsingRfcommWithServiceRecord(SNAME, MY_UUID);
        } catch (IOException e) {
            Log.e(TAG, "Socket's listen() method failed", e);
        }
        mmServerSocket = tmp;
    }

    public void run() {
        BluetoothSocket socket = null;
        // Keep listening until exception occurs or a socket is returned.
        while (true) {
            try {
                socket = mmServerSocket.accept();
            } catch (IOException e) {
                Log.e(TAG, "Socket's accept() method failed", e);
                break;
            }
            if (socket != null) {
                // A connection was accepted. Perform work associated with
                // the connection in a separate thread.
                manageMyConnectedSocket(socket);
                mmServerSocket.close();
                break;
            }
        }
    }

    // Closes the connect socket and causes the thread to finish.
    public void cancel() {
        try {
            mmServerSocket.close();
        } catch (IOException e) {
            Log.e(TAG, "Could not close the connect socket", e);
        }
    }
}
```

Pair with My device?
Bluetooth pairing code
222394

- Allow My device to access your contacts and call history
- CANCEL
- PAIR
private class ConnectThread extends Thread {
    private final BluetoothSocket mmSocket;
    private final BluetoothDevice mmDevice;

    public ConnectThread(BluetoothDevice device) {
        // Use a temporary object that is later assigned to mmSocket
        // because mmSocket is final.
        BluetoothSocket tmp = null;
        mmDevice = device;

        try {
            // Get a BluetoothSocket to connect with the given BluetoothDevice.
            // MY_UUID is the app's UUID string, also used in the server code.
            tmp = device.createRfcommSocketToServiceRecord(MY_UUID);
        } catch (IOException e) {
            Log.e(TAG, "Socket's create() method failed", e);
        }
        mmSocket = tmp;
    }

    public void run() {
        // Cancel discovery because it otherwise slows down the connection.
        mBluetoothAdapter.cancelDiscovery();

        try {
            // Connect to the remote device through the socket. This call blocks
            // until it succeeds or throws an exception.
            mmSocket.connect();
        } catch (IOException connectException) {
            // Unable to connect; close the socket and return.
            try {
                mmSocket.close();
            } catch (IOException closeException) {
                Log.e(TAG, "Could not close the client socket", closeException);
            }
            return;
        }

        // The connection attempt succeeded. Perform work associated with
        // the connection in a separate thread.
        manageMyConnectedSocket(mmSocket);
    }

    // Closes the client socket and causes the thread to finish.
    public void cancel() {
        try {
            mmSocket.close();
        } catch (IOException e) {
            Log.e(TAG, "Could not close the client socket", e);
        }
    }
}
How may you design a software framework supporting BT profiles?
Outline

- Admin
- Android
  - Platform overview
  - Basic concepts
  - Inter-thread: execution model with multiple threads
  - Inter-process: component composition
  - Inter-machine: network-wise composition
    - Service discovery
    - Make connections
      - Bluetooth
      - WiFi Direct also called WiFi P2P
WiFi Direct

- Initial support by a major vendor: Intel in their Centrino 2 platform in 2008
- Google announced Wi-Fi Direct support in Android 4.0 in October 2011
- Xbox in Xbox One in 2013
- Windows 8 on PC
- iOS has its own proprietary feature
WiFi Direct: Basic Ideas

- Essentially embeds a software access point ("Soft AP"), into any device that must support Direct.
  - The soft AP provides a version of Wi-Fi Protected Setup with its push-button or PIN-based setup
  - Only one device needs to be aware of WiFi Direct
- Allow devices to form groups

https://briolidz.wordpress.com/2012/01/10/wi-fi-protected-setup-wps/
Deployment

http://www.it.uc3m.es/pablo/papers/pdf/2012_camps_commag_wifidirect.pdf
Group Formation: Standard

See backup slides on other types of Group Formation
WPS Provisioning

See [https://en.wikipedia.org/wiki/Wi-Fi_Protected_Setup](https://en.wikipedia.org/wiki/Wi-Fi_Protected_Setup)
Discussion: Software API Design

Setup permission in Manifest

Create IntentFilter and Broadcast Receiver

- `intentFilter.addAction` with:
  - `WIFI_P2P_STATE_CHANGED_ACTION`
  - `WIFI_P2P_PEERS_CHANGED_ACTION`
  - `WIFI_P2P_CONNECTION_CHANGED_ACTION`
  - `WIFI_P2P_THIS_DEVICE_CHANGED_ACTION`

In life cycle

```java
/** register the BroadcastReceiver with the intent values to be matched */
@Override
public void onResume() {
    super.onResume();
    receiver = new WiFiDirectBroadcastReceiver(mManager, mChannel, this);
    registerReceiver(receiver, intentFilter);
}
```

```java
@Override
public void onPause() {
    super.onPause();
    unregisterReceiver(receiver);
}
```

Initiate

- Call `discoverPeers`
  ```java
  mManager.discoverPeers(mChannel, new WifiP2pManager.ActionListener()){
  ...
  ```
- Call `connect`
- `requestGroupInfo`
Outline

- Admin
- Android
  - Platform overview
  - Basic concepts
  - Inter-thread: execution model with multiple threads
  - Inter-process: component composition
  - Inter-machine: network-wise composition
    - Service discovery
    - Make connections
      - WiFi Direct
      - Cellular connect
Recall: GSM Logical Channels and Request

- Many link layers use a hybrid approach
  - Mobile device uses random access to request radio resource
  - The device holds the radio resource during a session

Diagram:
- Call setup from an MS
- RACH (request signaling channel) → BTS
- AGCH (assign signaling channel) ← MS
- SDCCH (request call setup) → BTS
- SDCCH message exchange
- SDCCH (assign TCH) ← MS
- Communication
RRC State Management in UMTS

Given the large overhead to set up radio resources, UMTS implements RRC state machine on mobile devices for data connection.

<table>
<thead>
<tr>
<th>Channel</th>
<th>Radio Power</th>
</tr>
</thead>
<tbody>
<tr>
<td>IDLE</td>
<td>Not allocated</td>
</tr>
<tr>
<td>Almost zero</td>
<td></td>
</tr>
<tr>
<td>CELL_FACH</td>
<td>Shared, Low Speed</td>
</tr>
<tr>
<td>Low</td>
<td></td>
</tr>
<tr>
<td>CELL_DCH</td>
<td>Dedicated, High Speed</td>
</tr>
<tr>
<td>High</td>
<td></td>
</tr>
</tbody>
</table>

https://developer.android.com/training/efficient-downloads/efficient-network-access.html

Courtesy: Erran Li.
RRC of a Large Commercial 3G Net

**DCH**: High Power State (high throughput and power consumption)

**FACH**: Low Power State (low throughput and power consumption)

**IDLE**: No radio resource allocated
RRC Effects on Device/Network

<table>
<thead>
<tr>
<th></th>
<th>FACH and DCH</th>
</tr>
</thead>
<tbody>
<tr>
<td><strong>Wasted Radio Energy</strong></td>
<td>34%</td>
</tr>
<tr>
<td><strong>Wasted Channel Occupation Time</strong></td>
<td>33%</td>
</tr>
</tbody>
</table>
Case Study: Pandora Streaming

Problem: High resource overhead of periodic audience measurements (every 1 min)
Recommendation: Delay transfers and batch them with delay-sensitive transfers
Case Study: Google Search

Search three key words.
ARO computes energy consumption for three phases

I: Input phase  S: Search phase  T: Tail Phase

Problem: High resource overhead of query suggestions and instant search
Recommendation: Balance between functionality and resource when battery is low
Summary

- App developers may not be aware of interactions with underlying network radio resource management

- See backup slides on Radio Resource Control (RRC) on 3G and LTE

- A good topic to think about as a part of your project
Backup Slides
Other Types of WiFi Group Formation Settings
Wifi Direct Group Formation: Autonomous Group

IEEE 802.11 Scan

Phase 1 & 2

DHCP Discover

DHCP Request

DHCP Offer

DHCP ACK

Discovery

WPS Provisioning

Address config.

Creates a P2P group and becomes GO
WiFi Direct Group Formation: Persistent
Eval: Formation Types Comparison
RRC State Transition in 3G and LTE
Radio Resource Control Setup for Data in 3G

RRC connection setup: ~ 1 sec + Radio Bearer Setup: ~ 1 sec


Source: Erran Li.
RRC State Transitions in LTE

- **RRC_CONNECTED**
  - Timer expiration

- **RRC_IDLE**
  - Data transfer

- **Continuous Reception**
- **Short DRX**
- **Long DRX**

- **DRX**

- $T_i$
- $T_{is}$
- $T_{tail}$
RRC State Transitions in LTE

RRC IDLE
- No radio resource allocated
- Low power state: 11.36mW average power
- Promotion delay from RRC_IDLE to RRC_CONNECTED: 260ms
RRC state transitions in LTE

RRC_CONNECTED
- Radio resource allocated
- Power state is a function of data rate:
  - 1060mW is the base power consumption
  - Up to 3300mW transmitting at full speed

RRC_IDLE
- Data transfer

Radio resource allocated
- Power state is a function of data rate:
  - 1060mW is the base power consumption
  - Up to 3300mW transmitting at full speed
RRC state transitions in LTE

Continuous Reception

Send/receive a packet
Promote to RRC_CONNECTED

Short DRX

Long DRX

Ti

T_{is}

T_{tail}

Reset T_{tail}

RRC_CONNECTED

Timer expiration

RRC_IDLE

Data transfer

Cellular Networks and Mobile Computing (COMS 6998-11)

Courtesy: Junxian Huang et al.
RRC state transitions in LTE

- **Continuous Reception**
- **Short DRX**
- **Long DRX**
- **RRC_CONNECTED**
- **RRC_IDLE**

- **Timer expiration**
- **Data transfer**

- **T_{tail} stops**
- **Demote to RRC_IDLE**

- **T_{tail} expires**

Courtesy: Junxian Huang et al.

Cellular Networks and Mobile Computing (COMS 6998-11)