AndroFleet: Testing WiFi Peer-to-Peer Mobile Apps in the Large

Lakhdar Meftah, Maria Gomez, Romain Rouvoy, Isabelle Chrisment

ANDROFLEET: Testing WiFi Peer-to-Peer Mobile Apps in the Large

Lakhdar Meftah [*], Maria Gomez [†], Romain Rouvoy [‡] and Isabelle Chrisment [§]
[*] Inria / University of Lille, France
[†] Saarland University, Germany
[‡] University of Lille / Inria / Institut Universitaire de France, France
[§] Telecom Nancy / Inria, France

***Abstract*—WiFi *P2P* allows mobile apps to connect to each other via WiFi without an intermediate access point. This communication mode is widely used by mobile apps to support interactions with one or more devices simultaneously. However, testing such P2P apps remains a challenge for app developers as *i)* existing testing frameworks lack support for WiFi P2P, and *ii)* WiFi P2P testing fails to scale when considering a deployment on more than two devices.**

In this paper, we therefore propose an acceptance testing framework, named ANDROFLEET, to automate testing of WiFi P2P mobile apps at scale. Beyond the capability of testing point-to-point interactions under various conditions, ANDROFLEET supports the deployment and the emulation of a fleet of mobile devices as part of an alpha testing phase in order to assess the robustness of a WiFi P2P app once deployed in the field. To validate ANDROFLEET, we demonstrate the detection of failing black-box acceptance tests for WiFi P2P apps and we capture the conditions under which such a mobile app can correctly work in the field. The demo video of ANDROFLEET is made available from https://youtu.be/gJ5_Ed7XL04.

I. INTRODUCTION

Mobile apps have brought the capability to deploy context-aware software systems, that go beyond desktop software, by adjusting their behavior to end-users' surrounding environment. In particular, WiFi *peer-to-peer* (WiFi P2P, also known as WiFi Direct) enables mobile devices to discover, connect, and transfer data to nearby devices via WiFi; without an intermediate access point as long as they comply with the WiFi Alliance's WiFi Direct certification [33]. This communication mode is now widely adopted by mobile apps with more than 1 billion of Wi-Fi Direct-enabled devices in 2014, and 3 billions expected by 2018[1] Some popular examples of WiFi P2P apps are multiplayer games, messaging, and social apps.

However, testing WiFi P2P apps remains a challenge for app developers [30] because current testing solutions lack support for the distributed nature of WiFi P2P interactions. In addition, current device emulators lack of an implementation for WiFi P2P communication stacks, thus developers are forced to use physical devices to test their WiFi P2P mobile apps. Testing scenarios typically involve more than two mobile devices, thus it is hard to scale to hundreds of mobile devices. Furthermore, reproducing behaviors becomes particularly difficult when evaluating stability, data integrity, peer failure and fault tolerance [32].

Developers then resort to alternative solutions to assess the expected behavior of their WiFi P2P mobile apps. For example, *log-based testing* [10] instruments apps and mines execution logs to detect unexpected behaviors. Given the potentially large number of mobile devices involved in WiFi P2P scenarios, log-based testing becomes unfeasible for developers who need to manually explore logs and assess a distributed behavior.

As existing automated testing methods are falling short on solutions to test WiFi P2P at large, we introduce ANDROFLEET, an acceptance testing framework for WiFi P2P Android apps. ANDROFLEET contributes the following features:

1) ANDROFLEET builds on a standard acceptance testing framework (*i.e.,* Cucumber[2]) to provide new primitives for describing contextual and distributed scenarios;
2) ANDROFLEET implements an emulated WiFi P2P communication stack to control the P2P interactions of mobile apps through emulated devices;
3) Finally, ANDROFLEET provides a large-scale deployment platform that can be used to assess the robustness of a WiFi P2P app prior to its deployment in the field, along an alpha testing phase. This feature collects insightful metrics on the execution and behavior of WiFi P2P apps.

ANDROFLEET is open-source and publicly available: https://github.com/m3ftah/androfleet.

The remainder of this paper is organized as follows. Section II describes the ANDROFLEET testing framework. Section III details the implementation of the different parts that compose the ANDROFLEET framework. Section IV reports on a case study that illustrates how ANDROFLEET supports developers to automate the testing of WiFi P2P apps. Section VI discusses the related work in this domain. Finally, Section VII concludes the paper.

II. TESTING PEER-TO-PEER MOBILE APPS

In this section, we present the ANDROFLEET framework to test WiFi P2P Android apps.

[1] http://blog.allion.com/2014/11/wi-fi-alliance-certified-wi-fi-direct-connections\ -provide-convenient-high-speed-network-services-direct-to-customers

[2] https://cucumber.io

Similar to any type of software system, developers need to ensure the correct behavior (*e.g.* absence of bugs and unexpected behavior) of WiFi P2P apps before deployment. The conception of peer-to-peer applications is complex as the communication scenarios between peers need to be specified and tested. To test such scenarios, there is therefore a need for a runtime environment, which is representative of a production deployment.

The goal of ANDROFLEET is to automate WiFi P2P *User Acceptance Testing*. ANDROFLEET enables developers to create and execute common WiFi P2P test scenarios. In particular, the framework provides two main features:

1) A DSL (*Domain-Specific Language*) to describe WiFi P2P testing scenarios, such as peer discovery and peer interactions;
2) A large-scale emulation platform that supports parallel testing on multiple devices as well as the WiFi P2P communications among them.

A. Describing Test Scenarios

To describe testing scenarios, ANDROFLEET provides a DSL to developers. Previous research have extended Calabash framework[3] to add some context-based testing [18], [17]. ANDROFLEET adopts a similar approach by extending Calabash to introduce a WiFi P2P specific vocabulary. We chose Calabash because it is multi-platform and delivers a rich and extensible syntax, based on Cucumber.

Figure 1 depicts the different ANDROFLEET testing directives that can be used to describe typical communication scenarios among peers in WiFi P2P apps.

In P2P systems different unexpected context events can happen, such as peer discovery, peer not in range or connection lost. However, some of these events can lead to failures of apps. ANDROFLEET lets developers simulate these contextual events that can emerge along an execution. Listing 1 describes an example of a test scenario, created with ANDROFLEET, to simulate P2P unexpected contextual events.

Listing 1: ANDROFLEET testing directives to manage WiFi P2P scenarios

```
Switch the device WiFi P2P state
Switch the device WiFi P2P permission
Switch the device WiFi P2P state
Send the available peer list has changed
    action
Change the device WiFi P2P name
Change the device WiFi P2P IP address
Change the device WiFi P2P role as group owner
Send the device WiFi P2P connection lost
    action
Get the device WiFi P2P state
```

B. Peer Discovery

Figure 2 (cf. top-right rectangle) shows an example of a testing scenario created with the ANDROFLEET DSL. This

[3]Calabash is a user acceptance testing framework: http://calaba.sh

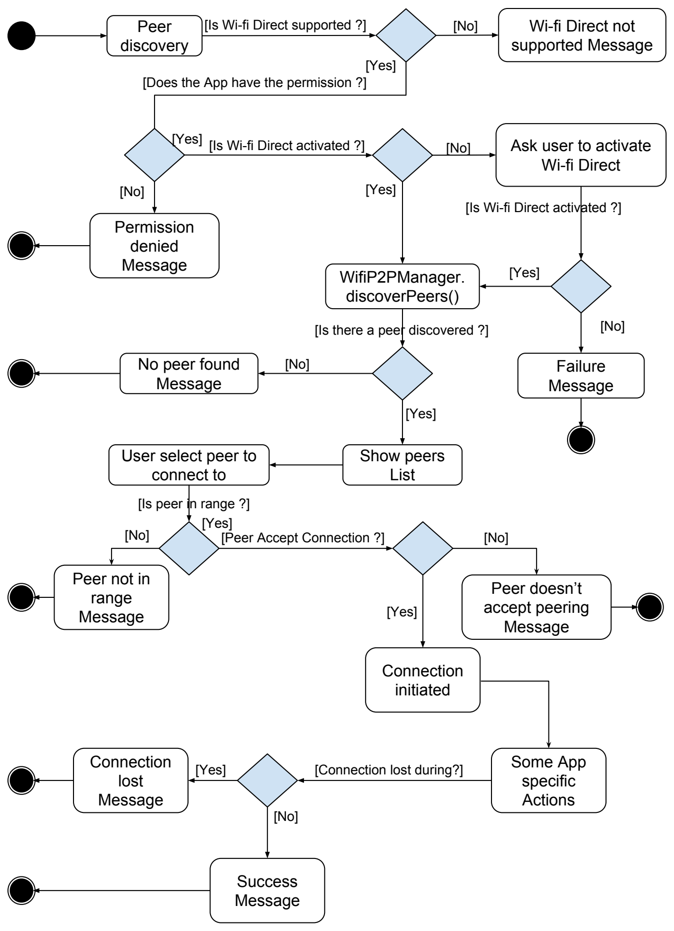

Peer discovery
[Is Wi-fi Direct supported ?]
[No]
Wi-fi Direct not supported Message
[Yes]
[Does the App have the permission ?]
[Yes]
[Is Wi-fi Direct activated ?]
[No]
Ask user to activate Wi-fi Direct
[No]
[Yes]
Permission denied Message
[Is Wi-fi Direct activated ?]
WifiP2PManager. discoverPeers()
[Yes]
[Is there a peer discovered ?]
[No]
No peer found Message
[No]
Failure Message
[Yes]
User select peer to connect to
Show peers List
[Is peer in range ?]
[Yes]
[No]
[Peer Accept Connection ?]
[No]
Peer not in range Message
[Yes]
Peer doesn't accept peering Message
Connection initiated
Connection lost Message
[Yes]
[Connection lost during?]
Some App specific Actions
[No]
Success Message

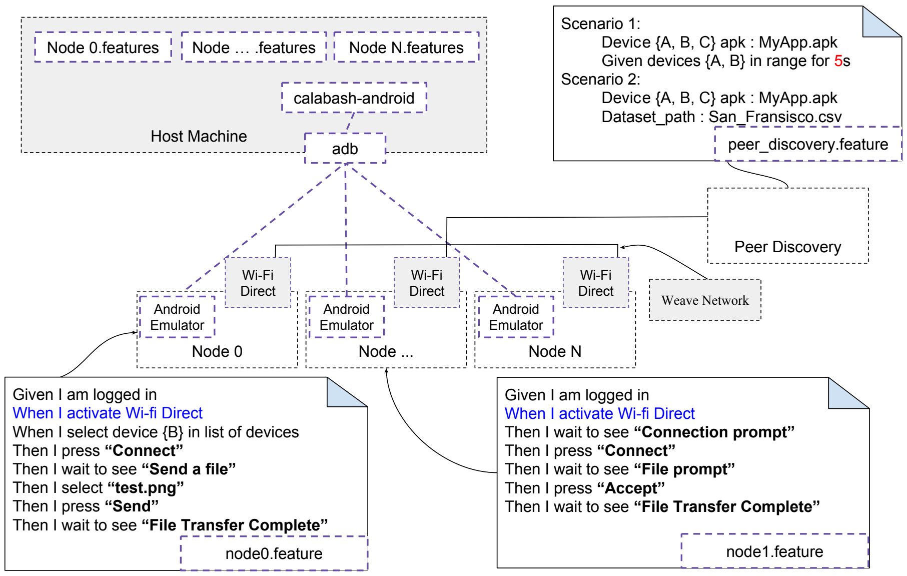
Fig. 1: Example of a communication scenario with WiFi-Direct

scenario tests the sharing of a file between two nodes (from Node 0 to Node 1) in a WiFi P2P File Sharing App. First, the developer has to specify the running conditions of the scenario using the `peer_discovery.feature` file, these conditions include:

- Number of participant nodes (*i.e.*, emulators)
  - Example: *Given the list of devices:* {*A, B, C*}
- Discovery configuration
  - Example: *Given devices* {*A, B*} *are in range for 5s*
- Apps installed on each node
  - Example: *MyApp.apk is installed on all devices*

Later, for each single node, the developer specifies the behavior to test. Figure 2 (cf. bottom rectangles) shows two examples of test scenarios in two nodes for the File Sharing app.

1) Discovery extension to Calabash: The apps asking for the list of nearby peers from the WiFi P2P hardware component will get this list from the $PeerDiscovery$. The list of available peers is configured using the `peer_discovery.feature` file.



Node 0.features
Node … .features
Node N.features
calabash-android
Host Machine
adb
Scenario 1:
Device {A, B, C} apk : MyApp.apk
Given devices {A, B} in range for 5s
Scenario 2:
Device {A, B, C} apk : MyApp.apk
Dataset_path : San_Fransisco.csv
peer_discovery.feature
Peer Discovery
Wi-Fi Direct
Android Emulator
Node 0
Node ...
Node N
Weave Network
Given I am logged in
When I activate Wi-fi Direct
When I select device {B} in list of devices
Then I press "Connect"
Then I wait to see "Send a file"
Then I select "test.png"
Then I press "Send"
Then I wait to see "File Transfer Complete"
node0.feature
Given I am logged in
When I activate Wi-fi Direct
Then I wait to see "Connection prompt"
Then I press "Connect"
Then I wait to see "File prompt"
Then I press "Accept"
Then I wait to see "File Transfer Complete"
node1.feature

Fig. 2: Overview of the ANDROFLEET framework

2) Opportunistic Peer Discovery: Some of the WiFi Direct Apps (emergency apps) use WiFi Direct to create a communication infrastructure [5], [14], [27], [12], [8], [15], [6], [11], [21], [28]. To prove the ability to support a communication infrastructure based on WiFi Direct, developers have to be able to use multiple GPS datasets that will help the Peer Discovery to emulate real-life nearby peers. Therefore, the developers can validate the scalability and the efficiency of the opportunistic communications for these communications infrastructures.

C. Running Test Scenarios

To run the specified test scenarios, the developer has to provide: (i) *test script scenarios* for each single node; (ii) *device profiles* (*e.g.*, OS version, hardware specification) to test the app; and (iii) `Peer_Discovery.features` as explained in Section II-B.

The scenarios are executed in parallel in multiple devices using Calabash-Android. After the execution of the scenarios, the developer can gather the test results, and improve the app accordingly and test all the scenarios again using ANDROFLEET.

III. IMPLEMENTATION DETAILS

The ANDROFLEET testing framework implements an extension of the Calabash testing framework to create test scenarios and execute them. ANDROFLEET is released as a Gradle plugin, which developers can add to run the tests inside Android Studio IDE. The ANDROFLEET gradle plugin executes the Calabash tests in parallel (in multiple devices) and collects the test results from all devices.

ANDROFLEET contains three parts: Device Emulator Nodes, a Peer Discovery Node, and a Host Machine. We describe the implementation of these three parts, respectively.

Device Nodes. ANDROFLEET uses Docker containers to deploy Android emulator nodes. We use the Docker technology to provide a lightweight and scalable testing environment. The different nodes communicate among them via Weave network.[4]

Peer Discovery. The Peer Discovery node is responsible for emulating WiFi Direct behavior in the Android emulator nodes and notify them about nearby devices. The communications between Device Emulator Nodes and the Peer Discovery node is achieved using the Akka framework, which supports large-scale deployment. In addition, the Peer Discovery node can use a GPS mobility dataset to compute the distances between devices and inform the devices when they are in range.

Host Machine. Finally, developers use a Host Machine to define and run testing scenarios. The Android emulators connect to the developer's machine via ADB (*Android Debug Bridge*).[5]

Peer-to-peer mobile apps are complex in nature, since they are based on a distributed environment. Thus, many aspects of the system need to be verified to ensure the correct behavior of the system under multiple circumstances (peer fails, peer refuses connexion, peer disappears, etc.).

ANDROFLEET implements an emulated version of the Android WiFi P2P API that can be deployed within an emulator (*e.g.*, GenyMotion) and controlled by the testing environment (*e.g.*, Calabash) to simulate the discovery of a nearby device and assess the user acceptance testing scenarios.

TABLE I: Testing scenarios for the $WiFiDirect$ app

#	Scenario	Expected behavior	Pass?
1	A simple image transferring scenario	Image is transferred	Yes
2	Transferring the image from the group Owner	Image is transferred	No
3	Deactivating WiFi P2P before transfer	Ask user to switch on WiFi P2P	Yes
4	Selecting a file instead of image	Open as dialog appears	Yes
5	Sending a second file in the same connection	The file is sent	No
6	Disconnect the second peer before starting the file transfer	Ask user to switch on WiFi P2P	No
7	Connecting to peer without launching discover request on it	The file is sent	Yes

IV. Case Study

In this section, we present a case study to illustrate how AndroFleet assists developers to automate the testing of WiFi P2P Android apps. As Application Under Test (AUT) we use $WiFiDirect$[6], an open-source app to transfer files between Android devices using WiFi Direct connection. The AUT has more than $10,000$ installs on the Google Play Store and its source code is available[7].

First of all, the developer needs to set up AndroFleet in the development IDE (*i.e.,* Android Studio). To do this, s/he uses the provided $Gradle\ Plugin$ which automatically mocks the Android WiFi P2P API used in the app.

Table I presents 7 relevant testing scenarios for the AUT. The developer describes the scenarios using the AndroFleet framework. Figure 3 shows the description of scenario #1, a simple image transferring from one device to another. In particular, listing 3a presents the sending scenario in one device; while 3b displays the receiving scenario in a different device.

Next, the developer runs the scenarios in the AndroFleet emulators. After execution, AndroFleet presents the testing results, *i.e.,* passing and failing tests. Figure 4 shows a screenshot of the testing results of the described case study. The screen displays the set of tests executed. Green and read colors mean passing and failing scenarios respectively. In the example, *Scenario 1* passed, while *Scenario 2* failed. Each scenario shows the sequential actions that were executed. If an action fails, AndroFleet shows error information and the reason of the failure. For example, in Scenario 2, the action 13 (*"Then I should see 'Connecting to' "*) fails. The reminder actions of the test are printed in blue, meaning that they were not executed since the test execution failed before. After analyzing the testing results, the developer can fix the app and re-run the tests until all scenarios successfully pass.

V. Discussion

Although the general idea of emulating hardware capabilities (e.g. GPS, sensors) for testing is not new, the novelty of AndroFleet resides in providing an emulated WiFi Direct environment for testing. Given the increasing popularity of P2P mobile apps, this feature is crucial because none of current testing frameworks support WiFi P2P.

AndroFleet reports the following advantages to the mobile development and testing communities:

1) Extend the capability of an Android testing framework to enable the control of peer-to-peer interactions among mobile apps via WiFi Direct.
2) Provide enriched vocabulary for testing mobile apps using WiFi Direct.
3) Automate parallel testing on multiple emulated devices as well as WiFi Direct communications among them.
4) Reproduce behaviors on failing test scenarios to help developers to fix the apps.

AndroFleet is specially practical for developers who lack access to a farm of real devices to test. But even when a crowd of real devices is available, current testing frameworks lack primitives to assess distributed behavior.

VI. Related Work

We summarize related work in the main research areas connected to this research: *testing mobile apps* and *testing peer-to-peer systems*. AndroFleet aims to fill the gap between these two areas.

A. Testing Mobile Apps

Mendez-porras *et. al.* [26] presented a systematic literature review on existing testing approaches for mobile apps. First, there is a bunch of approaches [34], [2], [19], [23], [1], [35] providing GUI testing of mobile apps. Similar to these approaches, AndroFleet also supports GUI testing. While the goal of previous approaches is to provide exploration strategies to maximize test coverage, AndroFleet provides additional primitives for testing properties specific to peer-to-peer systems (*e.g.*, connections, data interchange).

Another family of approaches have proposed cloud-based frameworks to test mobile apps [13], [22], [31]. AndroFleet also provides a cloud-based framework. In addition, AndroFleet provides multiple emulators, which can interact among them in order to test the behavior of a P2P system as a whole.

AndroFleet provides black-box testing to test WiFi P2P apps without requiring access to the source code. Previous approaches [20], [24], [36] propose also automatic black box testing of mobile apps.

Similar to AndroFleet, other approaches have extended the Calabash framework to support the testing of additional features, such as touch gestures [18] and sensors (*i.e.*, accelerometer, GPS, etc.) [17]. Similarly, AndroFleet extends

[4]https://www.weave.works/oss/net
[5]https://developer.android.com/studio/command-line/adb.html
[6]https://play.google.com/store/apps/details?id=anuj.wifidirect
[7]https://github.com/anuj7sharma/WiFiDIrectDemo

(a) Scenario send image file (Node 1)

```
Feature: File sharing feature
  Scenario: Simple image sharing — sending
  Given I activate WiFi Direct
  Given I wait up to 60 seconds for ”Available” to appear
  Given I don’t see the text ”Disconnect”

  When I take an image
  Then I press the menu key
  Then I press the text view ”Discover”
  Then I should see ”finding peers”
  Then I wait up to 60 seconds for ”N192168492” to appear
  Then I wait up to 60 seconds for ”CHOOSE FILE” to appear
  Then I press the ”CHOOSE FILE” button
  Then I take a screenshot
  Then I tap in 150 200
  Then I tap in 150 200
```

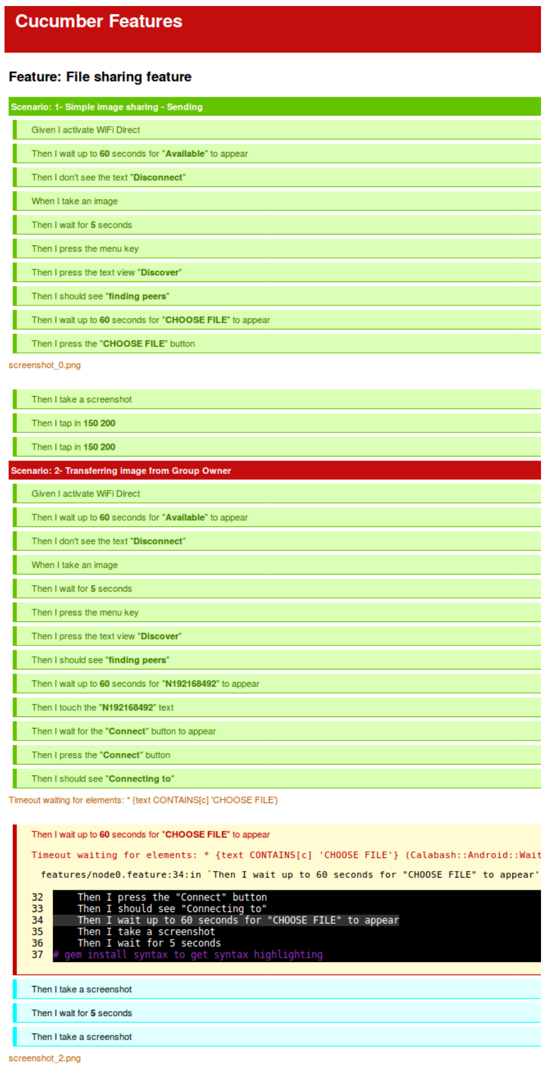
(b) Scenario receive image file (Node 2)

```
Feature: File sharing feature
  Scenario: Simple image sharing — receiving
  Given I activate WiFi Direct
  Given I wait up to 60 seconds for ”Available” to appear
  Given I don’t see the text ”Disconnect”

  When I take a screenshot
  Then I press the menu key
  Then I press the text view ”Discover”

  When I should see ”finding peers”
  Then I wait up to 60 seconds for ”N192168491” to appear
  Then I touch the ”N192168491” text
  Then I wait for the ”Connect” button to appear
  Then I press the ”Connect” button
  Then I should see ”Connecting to”
  Then I wait for 15 seconds
  Then I take a screenshot
```

Fig. 3: Description of sharing file scenario in the AUT


Cucumber Features

Feature: File sharing feature

Scenario: 1- Simple image sharing - Sending

Given I activate WiFi Direct

Then I wait up to **60** seconds for "**Available**" to appear

Then I don't see the text "**Disconnect**"

When I take an image

Then I wait for **5** seconds

Then I press the menu key

Then I press the text view "**Discover**"

Then I should see "**finding peers**"

Then I wait up to **60** seconds for "**CHOOSE FILE**" to appear

Then I press the "**CHOOSE FILE**" button

screenshot_0.png

Then I take a screenshot

Then I tap in **150 200**

Then I tap in **150 200**

Scenario: 2- Transferring image from Group Owner

Given I activate WiFi Direct

Then I wait up to **60** seconds for "**Available**" to appear

Then I don't see the text "**Disconnect**"

When I take an image

Then I wait for **5** seconds

Then I press the menu key

Then I press the text view "**Discover**"

Then I should see "**finding peers**"

Then I wait up to **60** seconds for "**N192168492**" to appear

Then I touch the "**N192168492**" text

Then I wait for the "**Connect**" button to appear

Then I press the "**Connect**" button

Then I should see "**Connecting to**"

Timeout waiting for elements: * {text CONTAINS[c] 'CHOOSE FILE'}

Then I wait up to **60** seconds for "**CHOOSE FILE**" to appear

```
Timeout waiting for elements: * {text CONTAINS[c] 'CHOOSE FILE'} (Calabash::Android::Wait

  features/node0.feature:34:in `Then I wait up to 60 seconds for "CHOOSE FILE" to appear'

32      Then I press the "Connect" button
33      Then I should see "Connecting to"
34      Then I wait up to 60 seconds for "CHOOSE FILE" to appear
35      Then I take a screenshot
36      Then I wait for 5 seconds
37  # gem install syntax to get syntax highlighting
```

Then I take a screenshot

Then I wait for **5** seconds

Then I take a screenshot

screenshot_2.png

Fig. 4: Testing results screenshot

Calabash with a different purpose: to support testing of peer-to-peer communications.

Despite the prolific research in this area, current testing frameworks for mobile apps lack support for peer-to-peer interactions.

B. Testing of Peer-to-Peer Systems

Almeida *et al.* [7] present a framework and a methodology for testing P2P applications. The framework is based on controlling nodes individually, allowing test cases to precisely control the volatility of nodes during their execution. Similarly, ANDROFLEET gives developers the ability to control the volatility of nodes during the execution of test scenarios.

Zhou *et al.* [37] propose a framework to test peer-to-peer multi-player games. In [29], the authors propose a model-based testing for large-scale distributed systems and peer-to-peer communications. Charaf *et al.* [9] propose an agent-based architecture for distributed system testing. Marroquin *et al.* [25] propose a testing approach based on test cases dependencies for communications protocols. Later, the same authors extend their approach to test distributed systems with test cases dependency architecture.

Butnaru *et al.* [4] propose a tool for measuring peer-to-peer platform performance using log messages to get feedback from different peers. Boldman *et al.* [3] explain in a patent a system and methods for testing peer-to-peer network application, by orchestrating unit testing between peers. Gorodetsky *et al.* [16] propose peer-to-peer emulation environment for testing mobile P2P agent applications.

Although testing P2P systems has been extensively studied by the research community, the testing of mobile P2P apps remains unexplored.

VII. CONCLUSION

This paper presents ANDROFLEET, an acceptance testing framework to automate the testing of WiFi P2P Android apps. ANDROFLEET provides a large-scale emulation platform which supports P2P interactions. ANDROFLEET enables the description of WiFi P2P testing scenarios (such as peer discovery and point-to-point interactions), as well as the execution of such tests in a crowd of emulated devices.

REFERENCES

[1] O. E. K. Aktouf, T. Zhang, J. Gao, and T. Uehara. Testing location-based function services for mobile applications. In *Proceedings - 9th IEEE International Symposium on Service-Oriented System Engineering, IEEE SOSE 2015*, volume 30, pages 308–314. IEEE, mar 2015.

[2] D. Amalfitano, A. R. Fasolino, P. Tramontana, and N. Amatucci. Considering context events in event-based testing of mobile applications. In *Proceedings - IEEE 6th International Conference on Software Testing, Verification and Validation Workshops, ICSTW 2013*, pages 126–133. IEEE, mar 2013.

[3] J. R. Boldman, U. W. Parks, R. L. Peruvemba, K. J. Prakash, and J. T. Wheeler. System and method for testing peer-to-peer network applications, 2007.

[4] B. Butnaru, F. Dragan, G. Gardarin, I. Manolescu, B. Nguyen, R. Pop, N. Preda, and L. Yeh. P2PTester: A tool for measuring P2P platform performance. In *Proceedings - International Conference on Data Engineering*, pages 1501–1502, 2007.

[5] D. Câmara, C. Bonnet, and F. Filali. Propagation of public safety warning messages: A delay tolerant network approach. In *IEEE Wireless Communications and Networking Conference, WCNC*, pages 1–6. IEEE, apr 2010.

[6] C. Cambra, S. Sendra, J. Lloret, and L. Parra. Ad hoc Network for Emergency Rescue System based on Unmanned Aerial Vehicles. *Network Protocols and Algorithms*, 7(4):72, jan 2016.

[7] E. C. de Almeida, G. Sunyé, Y. Le Traon, and P. Valduriez. Testing peer-to-peer systems. *Empirical Software Engineering*, 15(4):346–379, 2010.

[8] Y. Duan, C. Borgiattino, C. Casetti, C. F. Chiasserini, P. Giaccone, M. Ricca, F. Malabocchia, and M. Turolla. Wi-Fi Direct Multi-group Data Dissemination for Public Safety The Wi-Fi Direct Technology Multi-group Communication with Android devices. *Wtc*, pages 1–6, 2014.

[9] M. El, H. Charaf, M. Benattou, and S. Azzouzi. A JESS AGENT Based Architecture for Testing Distributed Systems. *JOURNAL OF INFORMATION SCIENCE AND ENGINEERING*, 30:1619–1634, 2014.

[10] A. Elyasov. Log-based testing. In *Proceedings of the 34th International Conference on Software Engineering*, ICSE '12, pages 1591–1594, Piscataway, NJ, USA, 2012. IEEE Press.

[11] M. D. Felice, L. Bedogni, and L. Bononi. The Emergency Direct Mobile App : Safety Message Dissemination over a Multi-Group Network of Smartphones using Wi-Fi Direct. In *Proceedings of the 14th ACM International Symposium on Mobility Management and Wireless Access*, pages 99–106, New York, New York, USA, 2016. ACM Press.

[12] G. Fodor, S. Parkvall, S. Sorrentino, P. Wallentin, Q. Lu, and N. Brahmi. Device-to-device communications for national security and public safety. *IEEE Access*, 2:1510–1520, 2014.

[13] J. Gao, W. T. Tsai, R. Paul, X. Bai, and T. Uehara. Mobile testing-as-a-service (MTaaS) - Infrastructures, issues, solutions and needs. In *Proceedings - 2014 IEEE 15th International Symposium on High-Assurance Systems Engineering, HASE 2014*, pages 158–167, 2014.

[14] S. M. George, W. Zhou, H. Chenji, M. Won, Y. O. Lee, A. Pazarloglou, R. Stoleru, and P. Barooah. DistressNet : A Wireless Ad Hoc and Sensor Network Architecture for Situation Management in Disaster Response. *Communications Magazine, IEEE*, 48(March):128–136, mar 2010.

[15] G. Gorbil. No way out: Emergency evacuation with no internet access. In *2015 IEEE International Conference on Pervasive Computing and Communication Workshops, PerCom Workshops 2015*, pages 505–511. IEEE, mar 2015.

[16] V. Gorodetsky, O. Karsaev, V. Samoylov, S. Serebryakov, S. Balandin, S. Leppanen, and M. Turunen. Virtual P2P environment for testing and evaluation of mobile P2P agents networks. In *Proceedings - The 2nd International Conference on Mobile Ubiquitous Computing, Systems, Services and Technologies, UBICOMM 2008*, pages 422–429. IEEE, sep 2008.

[17] T. Griebe and V. Gruhn. A model-based approach to test automation for context-aware mobile applications. *Proceedings of the 29th Annual ACM Symposium on Applied Computing - SAC '14*, pages 420–427, 2014.

[18] M. Hesenius, T. Griebe, S. Gries, and V. Gruhn. Automating UI tests for mobile applications with formal gesture descriptions. *MobileHCI 2014 - Proceedings of the 16th ACM International Conference on Human-Computer Interaction with Mobile Devices and Services*, pages 213–222, 2014.

[19] C. S. Jensen, M. R. Prasad, and A. Møller. Automated testing with targeted event sequence generation. *Proceedings of the 2013 International Symposium on Software Testing and Analysis - ISSTA 2013*, page 67, 2013.

[20] B. Jiang, X. Long, and X. Gao. MobileTest: A tool supporting automatic black box test for software on smart mobile devices. In *Proceedings - International Conference on Software Engineering*, pages 8–8. IEEE, may 2007.

[21] K. Koumidis, P. Kolios, C. Panayiotou, and G. Ellinas. ProximAid: Proximal adhoc networking to aid emergency response. In *Proceedings of the 2015 2nd International Conference on Information and Communication Technologies for Disaster Management, ICT-DM 2015*, pages 20–26. IEEE, nov 2016.

[22] C.-J. M. Liang, N. D. Lane, N. Brouwers, L. Zhang, B. F. Karlsson, H. Liu, Y. Liu, J. Tang, X. Shan, R. Chandra, and F. Zhao. Caiipa: Automated Large-scale Mobile App Testing through Contextual Fuzzing. In *MobiCom*, pages 519–530, New York, New York, USA, 2014. ACM Press.

[23] T. A. Majchrzak and M. Schulte. Context-Dependent Testing of Applications for Mobile Devices. *Open Journal of Web Technologies*, 2(1):27–39, 2015.

[24] K. Mao, M. Harman, and Y. Jia. Robotic Testing of Mobile Apps for Truly Black-Box Automation. *IEEE Software*, 34(2):11–16, mar 2017.

[25] A. Marroquin, D. Gonzalez, and S. Maag. A novel distributed testing approach based on test cases dependencies for communication protocols. In *Proceeding of the 2015 Research in Adaptive and Convergent Systems, RACS 2015*, pages 497–504, New York, New York, USA, 2015. ACM Press.

[26] A. Méndez-porras, C. Quesada-lópez, and M. Jenkins. Automated Testing of Mobile Applications : A Systematic Map and Review. *CIBSE 2015 - XVIII Ibero-American Conference on Software Engineering*, pages 195–208, 2015.

[27] O. Mokryn, D. Karmi, A. Elkayam, and T. Teller. Help Me: Opportunistic smart rescue application and system. In *2012 the 11th Annual Mediterranean Ad Hoc Networking Workshop, Med-Hoc-Net 2012*, pages 98–105. IEEE, jun 2012.

[28] M. Romano, T. Onorati, I. Aedo, and P. Diaz. Designing mobile applications for emergency response: Citizens acting as human sensors. *Sensors (Switzerland)*, 16(3):406, mar 2016.

[29] G. Sunyé, E. C. De Almeida, Y. Le Traon, B. Baudry, and J. M. Jézéquel. Model-based testing of global properties on large-scale distributed systems. *Information and Software Technology*, 56(7):749–762, 2014.

[30] G. Tassey. The economic impacts of inadequate infrastructure for software testing. Technical Report 7007, 2002.

[31] I. K. Villanes Rojas, S. Meireles, and A. C. Dias-neto. Cloud-Based Mobile App Testing Framework : Architecture , Implementation and Execution. In *Proceedings of the 1st Brazilian Symposium on Systematic and Automated Software Testing - SAST*, pages 1–10, New York, New York, USA, 2016. ACM Press.

[32] M. Wang, H. Wolf, M. Purvis, and M. Purvis. An Agent-based Collaborative Framework for Mobile Peer-to-Peer Applications. *System*, pages 132–144, 2004.

[33] Wi-FI Alliance. Wi-Fi Direct — Wi-Fi Alliance, 2015.

[34] S. Yu and S. Takada. External Event-Based Test Cases for Mobile Application. *Proceedings of the Eighth International C* Conference on Computer Science & Software Engineering*, pages 148–149, 2008.

[35] S. Yu and S. Takada. Mobile application test case generation focusing on external events. In *Proceedings of the 1st International Workshop on Mobile Development - Mobile! 2016*, pages 41–42, New York, New York, USA, 2016. ACM Press.

[36] T. Yumoto, T. Matsuodani, and K. Tsuda. A test analysis method for black box testing using AUT and fault knowledge. In *Procedia Computer Science*, volume 22, pages 551–560, 2013.

[37] Z. Zhou, H. Wang, J. Zhou, L. Tang, K. Li, W. Zheng, and M. Fang. Pigeon: A framework for testing peer-to-peer massively multiplayer online games over heterogeneous network. In *2006 3rd IEEE Consumer Communications and Networking Conference, CCNC 2006*, volume 2, pages 1028–1032, 2006.