Mobile Offload Computing

CSE 535: Mobile Computing

Abhik Kumar Dey
MSCS, CIDSE
Ira A. Fulton Schools of Engineering Arizona State University
akdey@*asu.edu*
1216907406

Rohith Eppepalli
CEN (CS), CIDSE
Ira A. Fulton Schools of Engineering Arizona State University
reppepal@asu.edu
1215350630

Sethu Manickam
MSCS, CIDSE
Ira A. Fulton Schools of Engineering Arizona State University
smanick4@asu.edu
1218452066

Sree Vashini Ravichandran
CEN (CS), CIDSE
Ira A. Fulton Schools of Engineering Arizona State University
sravic17@asu.edu
1217841794

***Abstract*- Given the rapid advancements of versatile applications and the expanding context of these apps, the computational demands of Smartphones and Wearable devices has grown significantly. To meet these computing demands, reliable and fast offloading techniques are being heavily researched. Our application has the underlying structure of offload computation performed through the P2P Wifi Direct standard. The application has a built in algorithm that selects the slave device by monitoring the resources available such as battery standby time to further increase the Quality of Service (QoS).**

I. Introduction

The advent of the Internet has brought in new waves of innovations and subsequent advancements in performance and computational efficiency of the portable devices. The best user experience can be achieved through increased real time performance of an application, but that puts a cap on the efficient use of the resources available as it requires higher end resources. Increased frequency of CPU due to complex computations ends up discharging the battery faster and heating the device. In order to increase the user-experience with reduced latency and exploitation of resources, this application uses P2P WiFi-direct to offload the data that needs to be processed. This approach significantly increased the real-time performance of the application due to parallel computation and the Master was consuming less power than that is required for local computation.

II. Author Keywords

Master, Slave, Wi-Fi manager, Peer-to-Peer (P2P), Broadcast Receiver.

III. Project Setup & Permissions

The architecture consists of one Master and N number of slaves. Using N number of slaves reduced the data transferring bandwidth as it must be equally split among N devices. Hence, 4 slave devices seem to be optimal for both latency and computational efficiency of a device [1].

1. Master (Android Device)
   OS: Android 7.0 and above
2. Slave (Android Device)
   OS: Android 7.0 and above

The android devices (Master and Slave) communicate with each other using bidirectional communication.

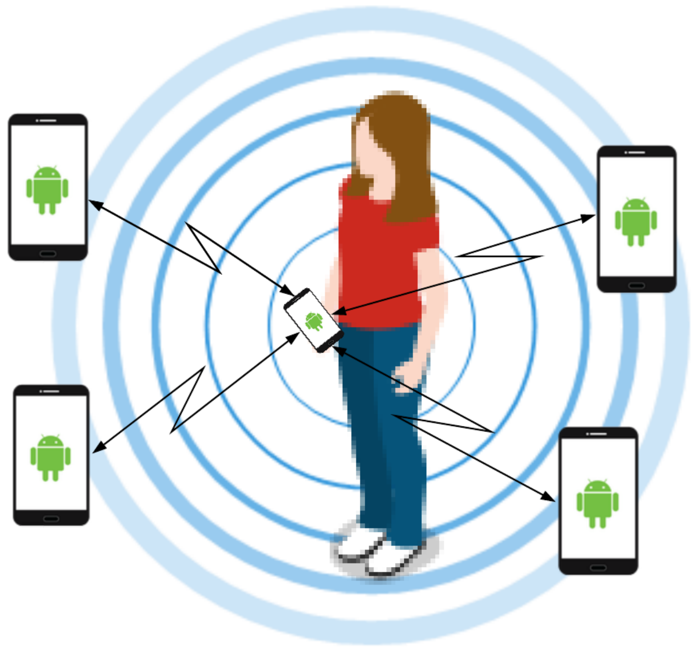


Figure 1: Architecture/Pipeline Diagram

IV. Proposed Approach

1. The Mobile Offloading application must be installed in the devices.
2. On starting the application, it scans for the peers/slaves that are available nearby proximity.
3. On discovery, Master send the request to connect
4. Upon accepting the request, the Master collects the battery and location details of the Peers/Slaves.
5. Based on the resource availability of the Slaves, Master decides the devices to which it can offload and creates separate socket threads for Slaves.
6. Two-way communication about the raw and processed data between the Master and Slaves is established.
7. After completing the data transmission, the Master disconnects with Slaves and closes the socket.

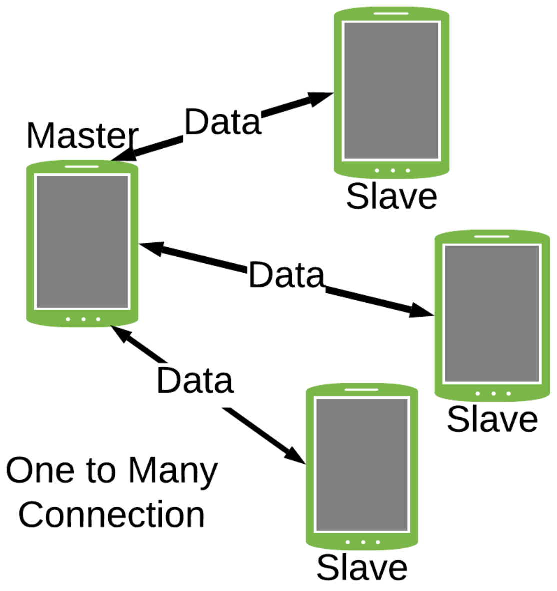


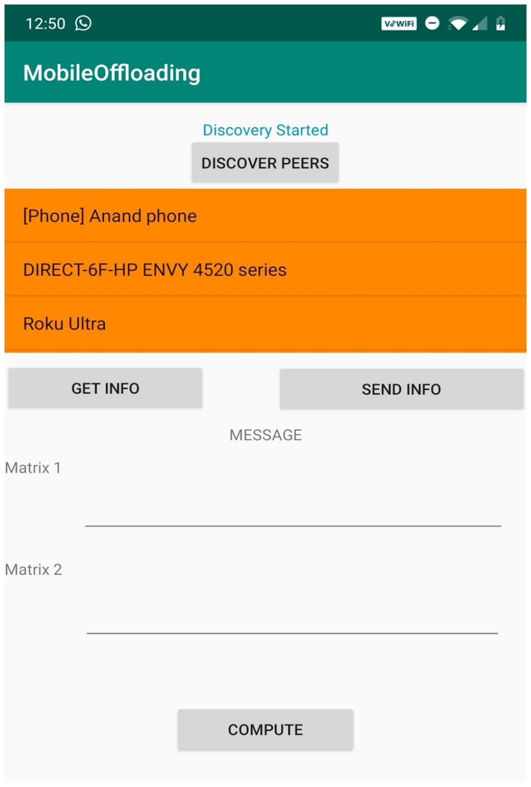
Figure 2: One to Many connections of WiFi P2p service

The application has both Master and Server functionality. The Wi-Fi manager service in android decides the Host/Master itself based on the comparison of the resources of the devices that are connected in the same Wi-Fi network. Then it forks the separate threads from the main UI to handle all the clients, this initiates the send and receive thread in the background that is responsible for the Master/Slave communication.

V. Implementation

The following are the tasks implemented for the project based on the guidelines provided:

1. *The Initial User Interface that is used to discover the peers/slaves available.*
   After opening the application, as we click on the discover button it discovers the peers/slaves that are available in the nearby proximity.


12:50

MobileOffloading

Discovery Started

DISCOVER PEERS

[Phone] Anand phone

DIRECT-6F-HP ENVY 4520 series

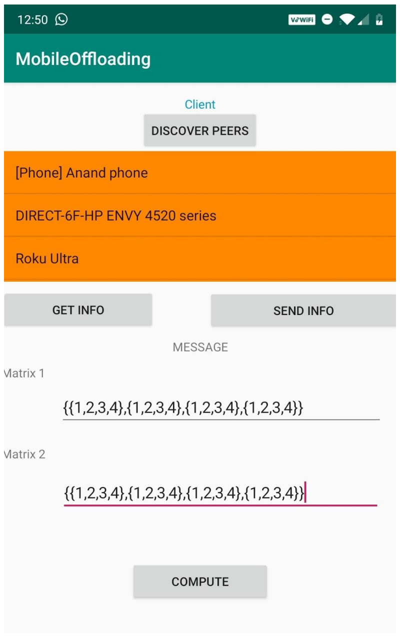
Roku Ultra

Figure 2: UI of Initial Screen

2. *Adaptive offloading algorithm based on the resource levels.*
   The minimum threshold that has been set for battery monitoring is 20%, if the battery level goes below 20% even after establishment of connection, the master disconnects the slave. This is a strict amendment of our proposal so that the application does not exploit the resources in process.

3. *Upon accepting the request, based on a selection algorithm, the master device starts to monitor the battery levels of the slaves.*
   This battery monitoring thread is initiated in the background as it has to continuously monitor the battery levels of slaves to perform effective offloading and efficient computation.

4. *User Interface screen to enter the matrix to be offloaded and computed*

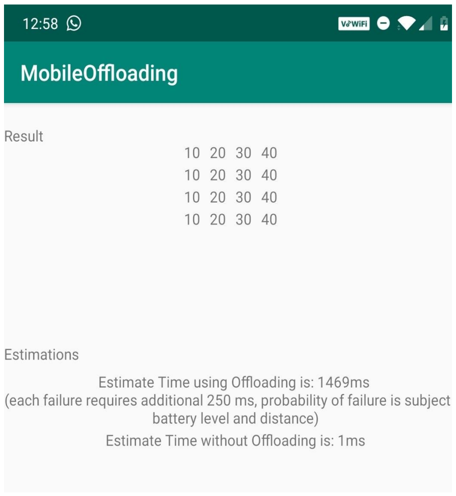


```
{{1,2,3,4},{1,2,3,4},{1,2,3,4},{1,2,3,4}}
{{1,2,3,4},{1,2,3,4},{1,2,3,4},{1,2,3,4}}
```

Figure 3: UI of the Input Screen

5. *Recovery Mechanism*
Master has a defined deadline to receive the computed results. If any slave fails to meet this deadline requirement during the computation or moves away from the proximity, then the Master looks for another slave that meets the minimum requirement and offloads the data to the new slave. The Master keeps track of all the task IDs that are being offloaded in a Hash Map, upon failure in receiving the results, the same task is offloaded again, but to a new slave.

6. *Performance Analysis*
Here, we observed that offloading simple processes was slower than expected. However, we must admit that in larger processing it will be impossible to do faster processing without offloading. The idea of using a Wi-Fi P2P network is to distribute the computation over a wider area in order to improve the resources available to compute the process. Hence, our system is expected to perform a lot better when using multiple devices while doing larger processing



```
10  20  30  40
10  20  30  40
10  20  30  40
10  20  30  40
```

Estimate Time using Offloading is: 1469ms
(each failure requires additional 250 ms, probability of failure is subject battery level and distance)
Estimate Time without Offloading is: 1ms

Figure 4: Matrix multiplication output with the estimation for offloading and local computation.

VI. Author Keywords

S.No.	Task Name	Assignee
1.	Master application to list all the available slaves.	Rohith
2	Requesting the available devices for participation.	Rohith
3	Adaptive offloading algorithm.	Rohith
4	Request to start battery monitoring of the chosen slaves.	Abhik
5	Enabling the accept feature in slaves.	Abhik
6	Continuous broadcast of the battery levels & GPS proximity.	Abhik
7	Inclusion of periodic monitoring in Slave.	Abhik
8	Matrix Computation.	Sethu
13	Failure Recovery Algorithm	Sethu
14	Estimation of execution time in local.	Sree Vashini

15	Estimation of execution time in distributed scenario without failure	Sree Vashini
16	Execution time estimation in distributed scenario with failure recovery	Sree Vashini
17	Power consumption estimation without distributed computation	Sree Vashini
18	Power consumption estimation with distributed computation	Rohith

VII. Limitations

The current implementation using WiFi direct is limited to 200m range. Future implementations can use a wireless access point to offload to all devices in a virtual private network. The app sends unencrypted data to offloadee devices, this is alright only as long as the network and devices are trusted. Future implementations can take into account privacy concerns and use an encryption scheme.

VIII. Conclusion

Although our implementation shows slower computation for offloading mechanism, we must also note that it is a simple 4x4 array multiplication. Here, the offloading time is much higher than the computation time. In cases, where the computation is massive, our application will be far more effective as computation power is massively compromised in a local system. But having multiple peer devices can help us in multiplying the computation power several fold. Thus, to conclude, our implementation of a Wi-Fi P2P system, although seems to perform poorly while processing 4x4 matrices, is expected to perform a lot better and consume optimum power while dealing with massive processing especially in highly constrained environments.

IX. Acknowledgment

We sincerely thank Professor Ayan Banerjee for giving us an opportunity to work on this project and guiding us throughout the project. This project helped us get hands on experience with mobile offloading computation framework. Throughout the project, we were able to learn and research various new concepts that made our understanding better about the computation and energy efficiency of a portable smart device.

X. References

[1] Golkarifard, Morteza, Ji Yang, Zhanpeng Huang, Ali Movaghar, and Pan Hui. "Dandelion: A unified code offloading system for wearable computing." *IEEE Transactions on Mobile Computing* 18, no. 3 (2018): 546-559.

[2] Wi-FiP2p documentation provided by Android https://developer.android.com/reference/android/net/wifi/p2p/package-summary