COEN 241 – Cloud Computing

Project: Off-Loading Algorithm on Simulated Mobile Edge Computing

By

Liang Xia, Weiben Zhang, Xuemei Wei, Nuo Dou
Date: June 12, 2017

Prof. Dr Ming-Hwa Wang

Preface

Since the invention of iPhone and Android mobile devices, smartphones have changed people's lifestyles enormously. Correspondingly, more and more mobile applications are developed, such as face recognition, interactive games, natural language processing, to fulfill all the aspects of demands. However, as these applications become more and more complicated, the conflicts between mobile devices' limited computation capacity and resource hungry applications are more and more intense. Thus, to solve the conflicts, people come up with the cloud service for mobile devices.

List of Figures:

Figure 1: Relation among Cloud Computing, Mobile Cloud Computing, Mobile Crowd Computing 5
Figure 2: Workflow of offloading tasks 12
Figure 3: Class Design 14
Figure 4: Initialize customized tasks 15
Figure 5: Start running jobs 15
Figure 6: Choosing a channel randomly 15
Figure 7: Offloading decisions 15
Figure 8: Process tasks on base station 15
Figure 9: cloud server processes tasks 15
Figure 10: Results in the log file 16
Figure 11: Energy and Time Saved under Different Battery Level with 30MB Data of EMU 17
Figure 12: Energy and Time Saved under Different Data Sizes with 75% Battery Level of EMU 17
Figure 13: Average Energy and Time Saved under Different Number of Devices of Dynamic Algorithm 18
Figure 14: Dynamics of system-wide computation overhead 18
Figure 15: Execution Time of DPR for a different number of tasks 19
Figure 16: Program flowchart 21

Table of Contents

1 Introduction 4

1.1 Objective 4
1.2 What is the problem 4
1.3 Why this is a project related to this class 4
1.4 Why other approach is no good and your approach is better 5

1.5 Area or scope of investigation 5

2 Theoretical Bases And Literature Review 5

2.1 Definition of the problem 5
2.2 Theoretical background of the problem 6
2.3 Related research to solve the problem 6
2.4 Advantage/disadvantage of those research 6
2.5 Your solution to solve this problem 6
2.6 Where your solution different from others 7
2.7 Why your solution is better 7

3 Hypothesis 7

4 Methodology 7

4.1 How to generate/collect input data 7
4.2 How to solve the problem 7
4.3 Algorithm design 7
4.4 Language used 8
4.5 Tools used 8
4.6 How to generate output 8
4.7 How to test against hypothesis 8

5 Implementation 8

5.1 Code (refer programming requirements) 8
5.2 Design Document and Flowchart 12

6 Data Analysis and Discussion 14

6.1 Output Generation 14
6.2 Abnormal Case Explanation 16
6.3 Output Analysis 16
6.4 Compare Output against Hypothesis 18

7 Conclusions and Recommendations 19

7.1 Summary and Conclusions 19
7.2 Recommendations for Future Studies 19

8 Bibliography 20

9 Appendices 20

1. Introduction

1.1 Objective:

Though mobile-edge cloud computing helps solve the conflicts, it cannot be randomly applied. Before offloading local computation tasks from mobile devices to the cloud, there should be certain algorithms to evaluate the performance, to determine whether offloading tasks to cloud is better than computing locally. In this project, we mainly focus on these evaluation algorithms which are used for making offloading decisions. We firstly read papers, to explore potential algorithms; then we simulate the situation of the mobile-edge cloud computing, like wireless stations and the cloud; after this, we apply the algorithms from the papers we selected; finally, we compare these algorithms against with each other, to have a performance analysis. By doing so, we can better understand the problems of mobile-edge cloud computing and the common solutions.

1.2 What is the problem:

The tension between resource-hungry applications and resource-constrained mobile devices hence poses the emerging of mobile-edge cloud computing.

- Computation capability: due to the physical constraint, mobile devices can only provide limited computation resources.
- Short battery life: the battery cannot last very long especially when performing tasks.
- Modern mobile applications are more and more resource-hungry which is too costly for mobile devices.

1.3 Why this is a project related to this class?

The class is about Cloud Computing and Mobile Edge Computing is part of Cloud Computing. Here we give the relationship among Cloud Computing, Mobile Cloud Computing and Mobile Crowd Computing(Mobile Edge Computing) in Figure 1. The difference between Mobile cloud Computing and Cloud Computing is that Mobile Cloud Computing only caters to mobile clients while Cloud Computing caters to both mobile and stationary clients. The difference between Mobile Edge Computing and Mobile Cloud Computing is Mobile Edge Computing only moves storage /computation to external mobile resources.

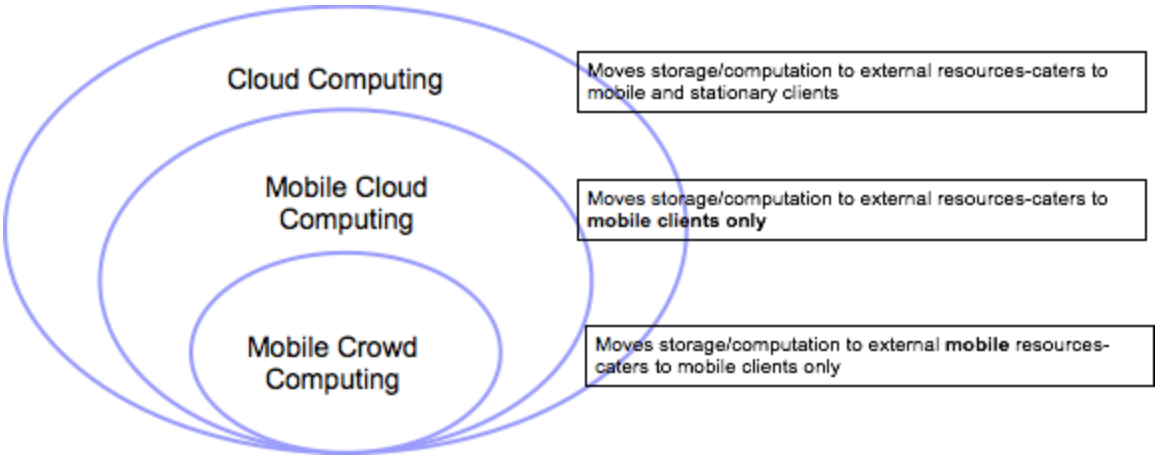


Figure 1 Relation among Cloud Computing, Mobile Cloud Computing, Mobile Crowd Computing

1.4 Why other approach is no good and your approach is better?

There are two other potential models for mobile cloud computing. One is to offload tasks to remote public cloud, another one is named as cloudlet based mobile cloud computing. The problems for Offload Tasks to Remote Public Cloud are the hard-to-reduced long latency. The problem for Cloudlet Based Mobile Cloud Computing is that it cannot provide accessible cloud service to users at anytime and anywhere, also the computation power of its cloud service provider may also be limited due to the physical constraints. By using cellular wireless network and fiber link, the MEC solves both the latency and coverage problems.

For the algorithm, the most well-known algorithm is called Cross Entropy Based Centralized Optimization. This algorithm works the best when all the detailed information of users are known by the wireless stations. However, to gain the information, the cost of the overhead of data commuting may be too high. Whereas, the algorithms we are going to apply in this project can make decisions based on limited information. Thus, in real situation, the algorithms applied in this project are more applicable.

1.5 Area or scope of investigation:

We mainly focus on two algorithms. One is designed based on a static-scenario which assumes the number of users remains unchanged during an off-loading period. The other is designed based on the Dynamic Programming, which is an optimization technology that transforming complex problems into a sequence of general problems, with Randomization strategy. Also, before implementing the algorithm, we have to simulate the real situations. Additionally, energy saving, pervasive computing and base station technologies are included in investigation.

2. Theoretical Bases And Literature Review

2.1 Definition of the problem

The problem includes the simulating of the real mobile-edge cloud computing scenario which is required before implementing problem, implementing the offloading algorithms basing on the he chosen papers and analyzing performance by specially designed input and configured parameters.

2.2 Theoretical background of the problem

As stated in the preface part, offloading tasks to cloud may or may not be beneficial, which depends on the cost evaluation results on both offloading tasks to cloud and completed computing locally. Thus, before sending tasks out, the evaluation should be done. The problem includes: how to properly define the system model to modulate actual situation into mathematical expressions; how to subtract formulate from information gained in last step; how to apply the formulations into real situation to analyze; and how to analyze performance. The papers that will be studied in this project solve the questions above perfectly, which provides solid background.

2.3 Related research to solve the problem

The related research covers a very large field, including studies on energy saving issues, pervasive computing issues and even on wireless station issues, etc, The authors of two main papers that are studied in this project read through those related research detailed, and finally provide comprehensive algorithms.

2.4 Advantage/disadvantage of those research

In the paper Efficient Multi-User Computation Offloading for Mobile-Edge Cloud Computing, the authors proposed a very efficient algorithm. It provides better performance than CE Based Centralized Offloading algorithm if we take the consumption of parameters exchange into account. However, the disadvantage of this algorithm is that it is designed basing on a static scenario which assumes the number of users are never changed. Thus, this algorithm may fail to work properly when the number of users fluctuates.

In the paper *A Dynamic Programming Offloading Algorithm using Biased Randomization*, the authors propose a self-adjustable algorithm which can find near-optimal solutions quickly. However, since the authors assume and analyze basing on WiFi network connection, the performance may vary when the mobile devices are under 3G/4G/LTE networks, of which situations the consumption of both energy and time may be higher.

2.5 Your solution to solve this problem

The first thing we are going to do is to simulate the real situation of mobile-edge cloud computing. This step can be achieved by implementing server-client application. The server is used to simulate wireless station and the cloud, and the client is used to simulate the mobile devices. The second step is to implement the algorithms. We plan to implement two algorithms independently, then, if possible, we may combine these two algorithms together and implement it. Finally, we give sample inputs and parameters to test and analyze performance.

2.6 Where your solution different from others

For the algorithm part, our Nash Equilibrium Based Algorithm is a mobile device making decision model, while most other algorithms are centralized model which means Base Station will make a decision for mobiles to choose which task should be offloaded to cloud, which would stay in local to compute.

For the implementation part, the main difference is where and whom the algorithms are going to applied to. In the original papers, the author apply and analyze algorithms on actual mobile devices and wireless stations. However, considering the lack of knowledge on mobile part, we can only achieve it by simulated server-client application. This may cause the results different with the papers', but should not be a very big difference.

2.7 Why your solution is better

For the algorithm part, our algorithm will combine the advantages from the Nash Equilibrium Algorithm and Biased Randomization Algorithm. First Nash Equilibrium Algorithm have a good performance and high efficiency with a fixed number of users in the cloud while the biased randomization algorithm help to fit to the actual situation perfectly when mobile users join and leave the cloud dynamically.

For the implementation part, the biggest benefit of our methodology also relates to the way we simulate the real situation. By simulation, it enables us to easily configure the parameters of both mobile devices and wireless station, which helps to know in what case the performance could be best. It may also be used to guide the actual design of products.

3. Hypothesis

The mobile devices are able to make decisions about offloading by only knowing limited information. The performance of applied algorithms should be much better than both the performance of locally computing and plain cloud computing, which randomly chooses channels to offload tasks, but it may be lower than CE Based Centralized Offloading algorithm. The algorithms may be unable to work properly when the number of users is changed during offloading period.

4. Methodology

4.1 How to generate/collect input data

Running client with a parameter as a weight value to simulate off-loading tasks.

4.2 How to solve the problem

Simulating the real situation, implementing the algorithms and analyzing the performance.

4.3 Algorithm design

Referring to the papers.

4.4 Language used

Java

4.5 Tools used

Eclipse IDE

4.6 How to generate output

Printing logs in terminals on both clients(mobile devices) and server(base station).

4.7 How to test against hypothesis

- Multiple users request to off-loading tasks simultaneously.
- User requests to off-loading tasks with different weight values. The higher the weight value is, the higher cost the task has.

5. Implementation

5.1. Code (refer programming requirements)

- Code of EMU's offloading algorithm

```
/**
 * Use Efficient Multi-user Algorithm to make offload decision
 * @param device : current mobile device
 * @param port : randomized port number which was used to connect to wireless station
 */
public void offloadingDecisionByMEC(MobileDevice device, int port) {
        // get the port which provides best performance
        int bestPort = calculateOverheadForMEC(port, device);
        // set the port
        device.setTargetPort(bestPort);
        // decide offloading weight according to the status of current device
        setOffloadingWeight(device);
}
```

- Code of dynamic offloading algorithm

```
        /**
 * Use the Dynamic Algorithm to make offload decision
 * @param device : current mobile device
 */
public void offloadingDecisionByDynamic(MobileDevice device) {
        // the matrix is a list of integer which stores decisons for all the tasks
        // to make it consistent, if the device's id is bigger than current list's size,
        // the previous not made decisions should be filled as -1
```

```
        if( device.getId() >= matrix.size() ) {
                for( int i = matrix.size(); i <= device.getId(); ++i) {
                        matrix.add(-1);                 // -1 indicates the decision for task i has
not made
                }
        }
        // random(): a 0 - 1 generator which is used to
        // generate decisions according to decision history;
        //
        if( matrix.get(device.getId()) == -1) {
                matrix.set(device.getId(), random());
        }
        // use the dynamic algorithm to make decision
        if( matrix.get(device.getId()) == 0 && !isBeneficial(device) ) {
                // offloading is not applicable, set offloading weight to 0
                device.setOffloadWeight(0);
        }
}
```

- Code that helps dynamic offloading algorithm to find best channel

```
/**
 * To see whether a decision is beneficial for dynamic algorithm
 * @param device
 * @return
 */
public boolean isBeneficial(MobileDevice device) {
        // connectionInfo of the wireless station
        Hashtable<Integer,Integer> connectionInfo =
WirelessStation.getConnectionInfo();
        //      get the total time span to complete the task
        double localTimeOverhead = device.getCompletelyLocalySpan();
        //      get the total energy consumption
        double localEnergyOverhead = localTimeOverhead/2;
        double minCloudOverhead = Double.MAX_VALUE;
        int transferTimeSpan = -1;
        int bestPort = -1;
        // compare each channel's overhead with locally computing's, and to find
        // a best channel which gives best performance
        for( int port : WirelessStation.getPorts()) {
                int lossFactor = connectionInfo.get(port) * 2;
```

```
                double curOverhead = 6
                                +  ( (double) device.getDataSize() /
(double)WirelessStation.getBandWidth()) * lossFactor
                                +  ( (double) device.getDataSize() /
(double)WirelessStation.getBandWidth()) / 3;
                if( curOverhead < minCloudOverhead) {
                        minCloudOverhead = curOverhead;
                        transferTimeSpan = device.getDataSize() /
WirelessStation.getBandWidth() * lossFactor;
                        bestPort = port;
                }
        }
        if( minCloudOverhead < (localTimeOverhead + localEnergyOverhead)) {
                if( transferTimeSpan > 30) return false;
                device.setTargetPort(bestPort);
                device.setCloudTimeSpan(6);
                device.setTransferTimeSpan(transferTimeSpan ) ;
                device.setOffloadWeight(1);
                return true;
        }
        return false;
}
```

- Code that helps EMU algorithm to find the best channel

```
/***
 * calculate overhead for emu algorithm
 * @param connectionInfo
 * @param port
 * @param device
 * @return
 */
public int calculateOverheadForMEC(int port, MobileDevice device) {
        //      get current connection info
        int currentChannelConnectionNum =
WirelessStation.getConnectionInfo().get(port);
        int lossFactor = WirelessStation.getLossFactor();
        int bandWidth = WirelessStation.getBandWidth();
        // get current overhead
        double currentOverHead = calculateOverhead(lossFactor,
currentChannelConnectionNum, bandWidth);
```

```
        // get conditional overhead which represents the overhead to let the task use
another channel
        double conditionalOverhead =
calculateOverhead(lossFactor,currentChannelConnectionNum - 1, bandWidth);
        // calculate other channel's overhead and find a best one
        double[]  connections= new double[WirelessStation.getPorts().length - 1];
        double[]  lossFactors = new double[connections.length];
        double[]  overheads = new double[connections.length];
        int[] ports = new int[overheads.length];
        int index = 0;
        for( int otherPort : WirelessStation.getPorts() ) {
                if( otherPort == port) continue;
//                      overheads[index] =
WirelessStation.getConnectionInfo().get(otherPort);
                connections[index] = WirelessStation.getConnectionInfo().get(otherPort);
                lossFactors[index] = Math.pow(WirelessStation.getLossFactor(),
connections[index]);
                ports[index] = otherPort;
                currentOverHead += calculateOverhead(lossFactor,
connections[index],bandWidth);
                index++;
        }
        index = 0;
        for( int i = 0; i < overheads.length; ++i) {
                overheads[i] = conditionalOverhead + calculateOverhead(lossFactor,
connections[i]+1, bandWidth);
        }
        double minOverhead = currentOverHead;
        int bestPort = port;
        // get the best channel
//              System.out.println("initialOverhead:   "+currentOverHead);
        for( int i = 0; i < overheads.length; ++i) {
                if( minOverhead > overheads[i]) {
                        minOverhead = overheads[i];
                        bestPort = ports[i];
                }
        }
        return bestPort;
}
```

5.2. Design Document and Flowchart

- Workflow of offloading tasks

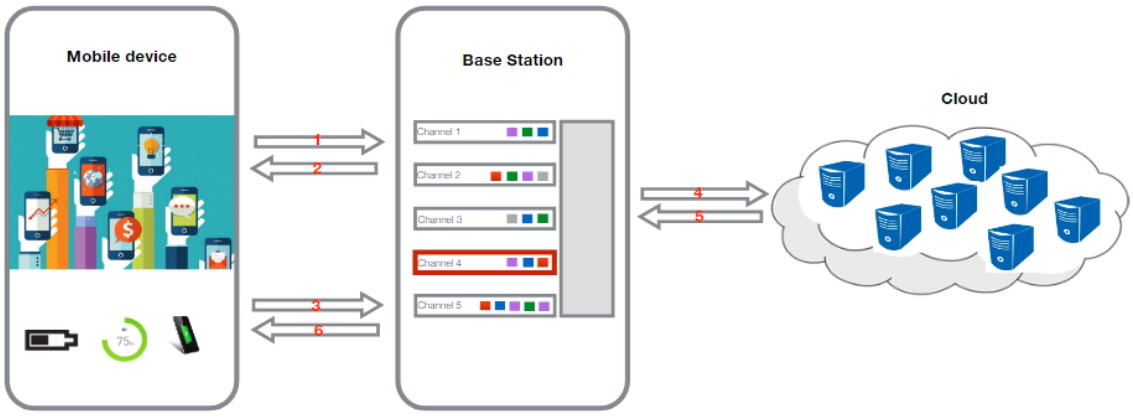


Figure 2 Workflow of offloading tasks

Step 1: Mobile device sends request to base station to get base station's information.

Step 2: Base station responses mobile device with information of availability and usage.

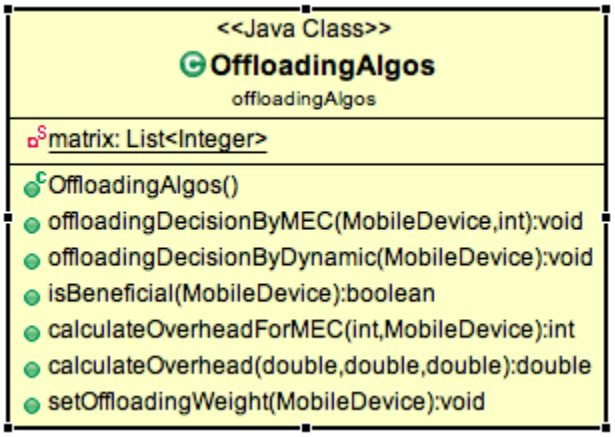
Step 3: Mobile device makes decision of offloading task based on the offloading algorithm. Then it sends the request to the base station with its choice.

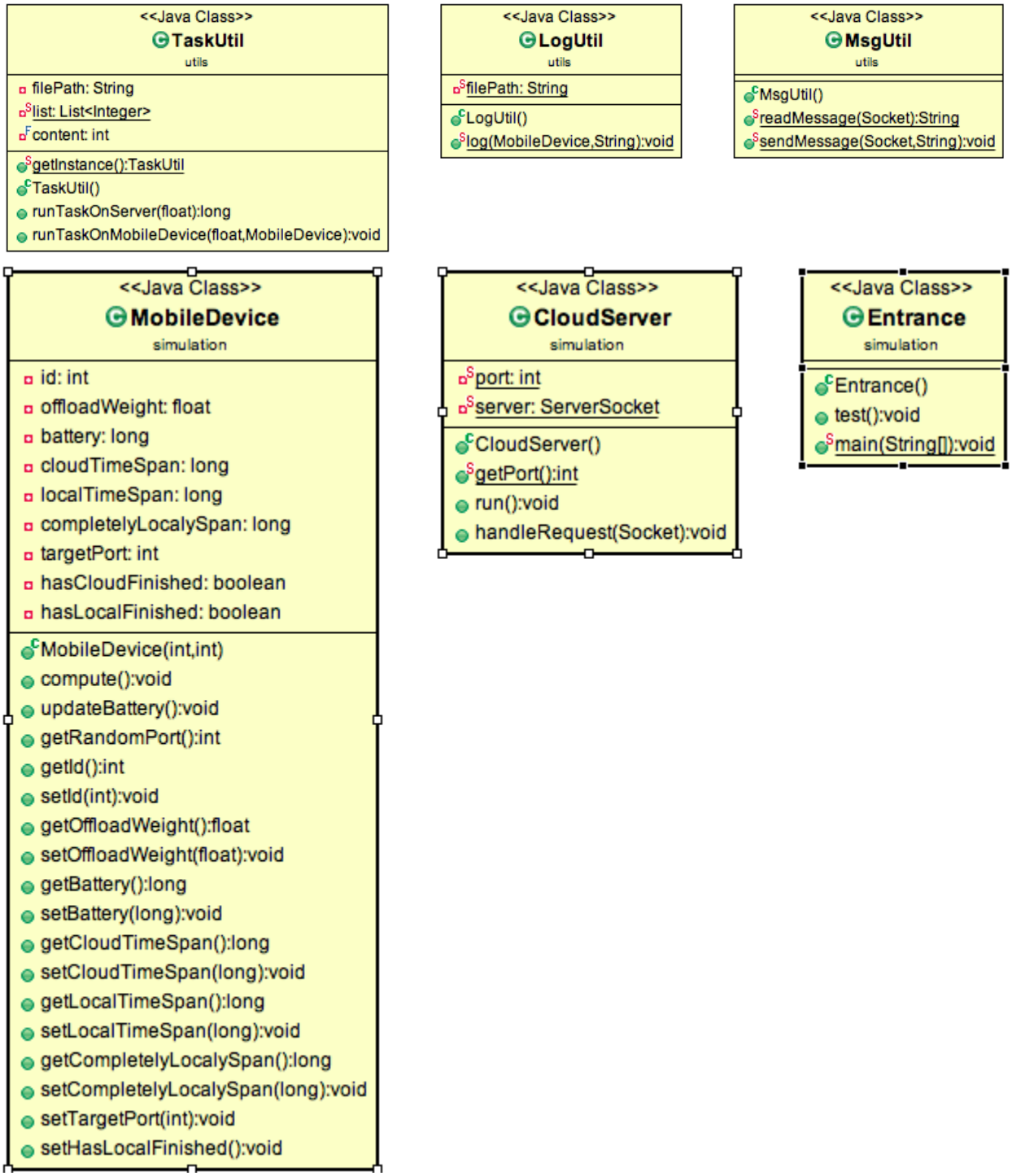
Step 4: Base station gets the request and offload the task in the targeted channel. Once previous tasks are finished in the same channel, the task will be sent to the cloud for calculation.

Step 5: Cloud responses the result back to the base station.

Step 6: Base station forwards the result back to the mobile device then removes the task from the channel.

- Class Design




<<Java Class>>
TaskUtil
utils
filePath: String
list: List<Integer>
content: int
getInstance():TaskUtil
TaskUtil()
runTaskOnServer(float):long
runTaskOnMobileDevice(float,MobileDevice):void
<<Java Class>>
LogUtil
utils
filePath: String
LogUtil()
log(MobileDevice,String):void
<<Java Class>>
MsgUtil
utils
MsgUtil()
readMessage(Socket):String
sendMessage(Socket,String):void
<<Java Class>>
MobileDevice
simulation
id: int
offloadWeight: float
battery: long
cloudTimeSpan: long
localTimeSpan: long
completelyLocalySpan: long
targetPort: int
hasCloudFinished: boolean
hasLocalFinished: boolean
MobileDevice(int,int)
compute():void
updateBattery():void
getRandomPort():int
getId():int
setId(int):void
getOffloadWeight():float
setOffloadWeight(float):void
getBattery():long
setBattery(long):void
getCloudTimeSpan():long
setCloudTimeSpan(long):void
getLocalTimeSpan():long
setLocalTimeSpan(long):void
getCompletelyLocalySpan():long
setCompletelyLocalySpan(long):void
setTargetPort(int):void
setHasLocalFinished():void
<<Java Class>>
CloudServer
simulation
port: int
server: ServerSocket
CloudServer()
getPort():int
run():void
handleRequest(Socket):void
<<Java Class>>
Entrance
simulation
Entrance()
test():void
main(String[]):void

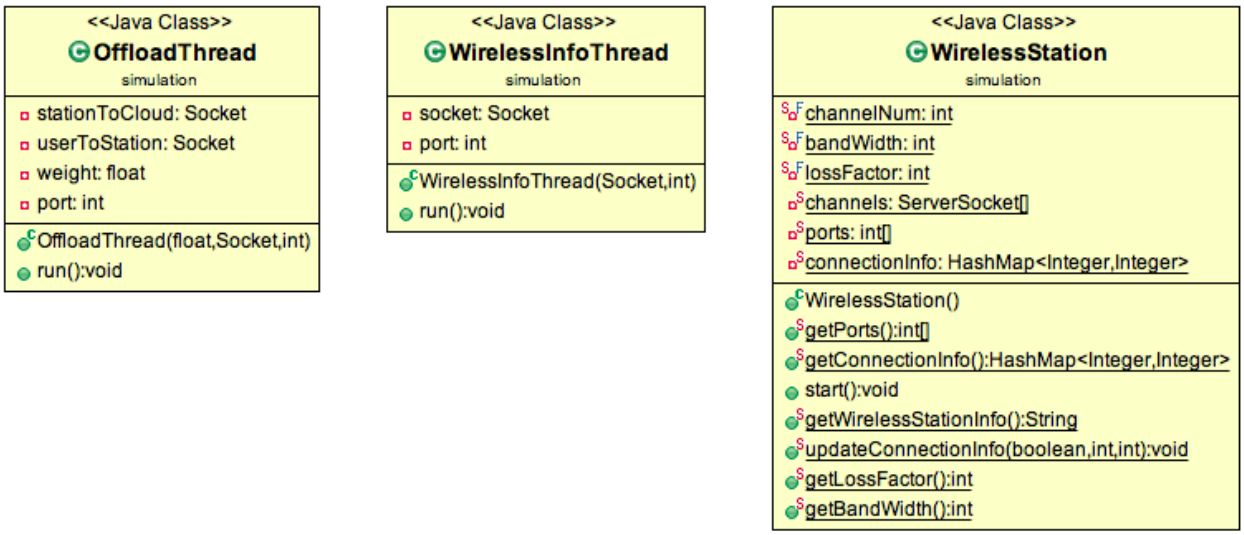


Figure 3 Class Design

6. Data Analysis and Discussion

6.1. Output Generation

- Wireless station, cloud server and mobile devices initialized

```
Wireless Station starts to run with 5 channels 62428 62429 62430 62431 62432
Cloud Server starts to run....
Input device number:5
Input battery level of device 1:(1~100 or -1 to randomize one)-1
Input data size of device 1:( 1~5 or -1 to randomize one)-1
Mobile Device 1 created with 13MB data and 71% battery life
Input battery level of device 2:(1~100 or -1 to randomize one)-1
Input data size of device 2:( 1~5 or -1 to randomize one)-1
Mobile Device 2 created with 41MB data and 43% battery life
Input battery level of device 3:(1~100 or -1 to randomize one)-1
Input data size of device 3:( 1~5 or -1 to randomize one)-1
Mobile Device 3 created with 56MB data and 5% battery life
Input battery level of device 4:(1~100 or -1 to randomize one)-1
Input data size of device 4:( 1~5 or -1 to randomize one)-1
Mobile Device 4 created with 22MB data and 72% battery life
Input battery level of device 5:(1~100 or -1 to randomize one)-1
Input data size of device 5:( 1~5 or -1 to randomize one)-1
Mobile Device 5 created with 39MB data and 87% battery life
```

Figure 4 Initialize customized tasks

- Devices start to run

```
Mobile device 1 starts to run by MEC offloading algorithm...
Mobile device 2 starts to run by MEC offloading algorithm...
Mobile device 3 starts to run by MEC offloading algorithm...
Mobile device 4 starts to run by MEC offloading algorithm...
Mobile device 5 starts to run by MEC offloading algorithm...
```

Figure 5 Start running jobs

- Devices randomly choose a channel at first

```
Mobile Device 3 communicates with wireless station on random selected channel 62432
Mobile Device 1 communicates with wireless station on random selected channel 62430
Mobile Device 2 communicates with wireless station on random selected channel 62431
Mobile Device 5 communicates with wireless station on random selected channel 62429
Mobile Device 4 communicates with wireless station on random selected channel 62428
```

Figure 6 Choosing a channel randomly

- Offloading decisions made

```
**********Accoding to MEC offloading algorithm, best channel changes from 62429 to 62428 on device5***
**********Accoding to MEC offloading algorithm, best channel changes from 62432 to 62429 on device3***
**********Accoding to MEC offloading algorithm, best channel changes from 62430 to 62432 on device1***
Device 3 decides to connect to channel 62429 which provides best performance
Device 1 decides to connect to channel 62432 which provides best performance
Device 5 decides to connect to channel 62428 which provides best performance
**********Accoding to MEC offloading algorithm, best channel changes from 62431 to 62432 on device2***
**********Accoding to MEC offloading algorithm, best channel changes from 62428 to 62432 on device4***
Device 2 decides to connect to channel 62432 which provides best performance
```

Figure 7 Offloading decisions

- Wireless station receives requests and transfer data to cloud server

```
Wireless station recieves request...
Device 3 offloads 0.9 of its total task to cloud
Device 4 decides to connect to channel 62432 which provides best performance
Device 1 offloads 0.6 of its total task to cloud
Wireless station recieves request...
Wireless station recieves request...
Device 2 offloads 0.7 of its total task to cloud
Device 4 offloads 0.6 of its total task to cloud
Wireless station recieves request...
```

Figure 8 Process tasks on base station

- Cloud server receives requests and starts to run

```
Cloud server starts computing
Cloud server starts computing
Cloud server starts computing
```

Figure 9 cloud server processes tasks

- An example of log information

```
----------------------------MEC---------------------------------
At the beginning, Device 1 has 71% of battery life and 13MB data to be transferred.
Device 1 has finished its task.
0.6 percentage of task has completed on cloud
0.4 percentage of task has completed locally
The transfer of data costs 2 seconds
The transfer of data uses 1% of total battery life
The transfer and local computation consumes 6% of total battery life
The task computes on the cloud uses 3 seconds
The task computes locally uses 11 seconds
Cloud computing helps reduce 19 seconds and 9% of total battery life
At the end, Device 1 has 65% of battery life remained
-------------------------------------------------------------------------
----------------------------Dynamic----------------------------------
At the beginning, Device 1 has 71% of battery life and 13MB data to be transferred.
Device 1 has finished its task.
1.0 percentage of task has completed on cloud
0.0 percentage of task has completed locally
The transfer of data costs 0 seconds
The transfer of data uses 1% of total battery life
The transfer and local computation consumes 1% of total battery life
The task computes on the cloud uses 6 seconds
The task computes locally uses 0 seconds
Cloud computing helps reduce 24 seconds and 14% of total battery life
At the end, Device 1 has 70% of battery life remained
-------------------------------------------------------------------------
```

Figure 10 Results in the log file

6.2. Abnormal Case Explanation

- Sometimes, if mobile devices randomly choose the channels which finish the task before getting the new tasks, randomly choosing channels will have higher performance compared to using offloading algorithms.

- Sometimes, the Dynamic Offloading Algorithm may let most of tasks to compute locally and only let several of tasks to be offloaded to the cloud. This happens because the Dynamic Offloading Algorithm doesn't care about the optimal solution for all the tasks, but just want to let the total execution time shorter than the constraint and minimize the total energy consumption.

- Sometimes, the EMU offloading Algorithm may give bad performance, this happens when the number of devices changes rapidly, which cause the situation of decision making period totally different with the situation when the task offloads to the cloud.

6.3. Output Analysis

- Energy and Time Saved under Different Battery Level with 30MB Data of EMU Offloading Algorithm

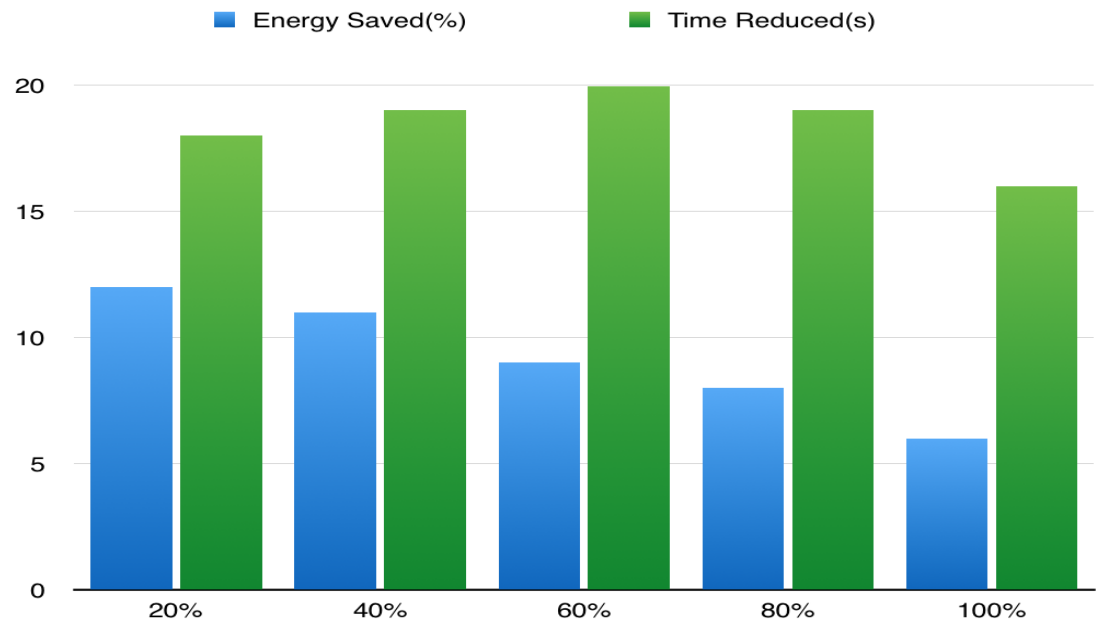


Figure 11 Energy and Time Saved under Different Battery Level with 30MB Data of EMU

- Energy and Time Saved under Different Data Sizes with 75% Battery Level of EMU Offloading Algorithm

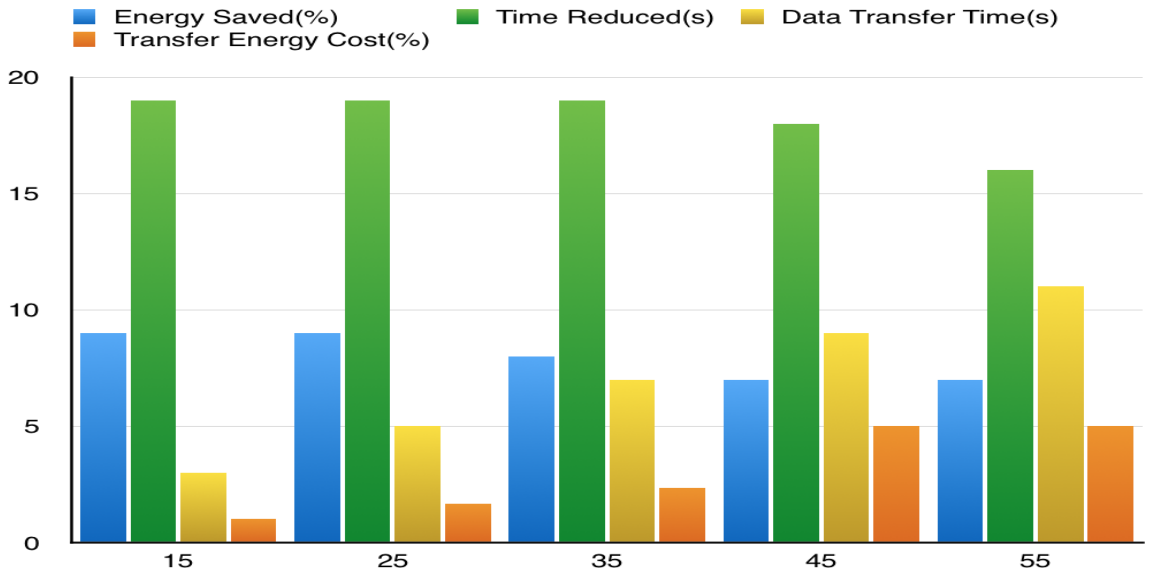


Figure 12 Energy and Time Saved under Different Data Sizes with 75% Battery Level of EMU

- Average Energy and Time Saved under Different Number of Devices of Dynamic Offloading Algorithm

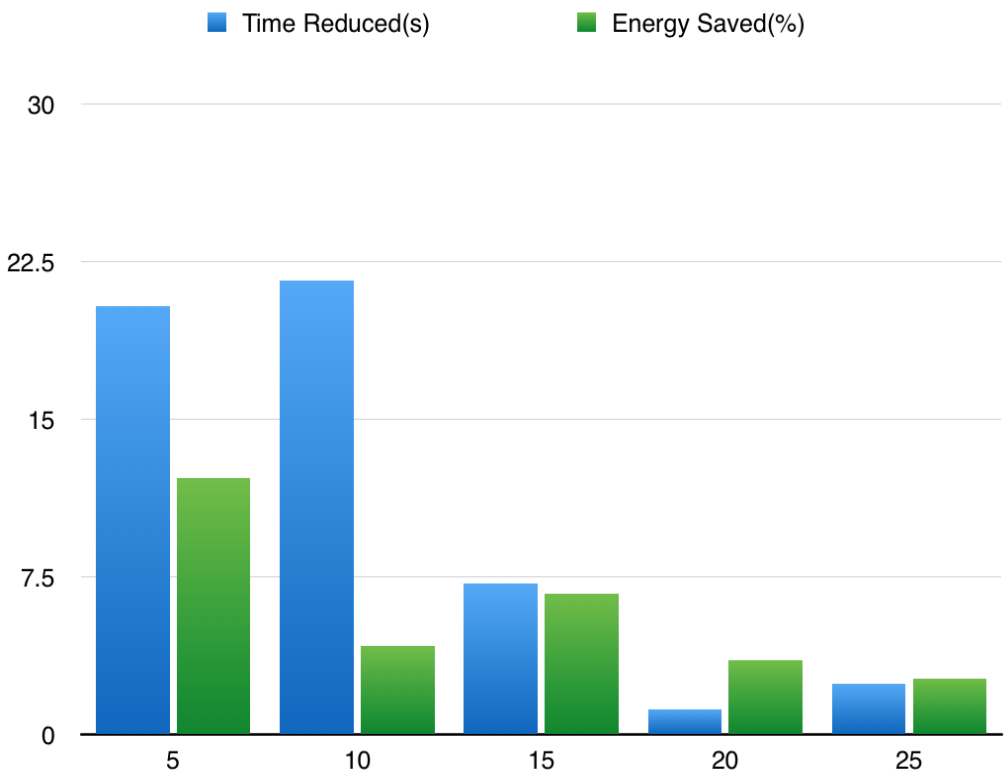


Figure 13 Average Energy and Time Saved under Different Number of Devices of Dynamic Algorithm

6.4. Compare Output against Hypothesis

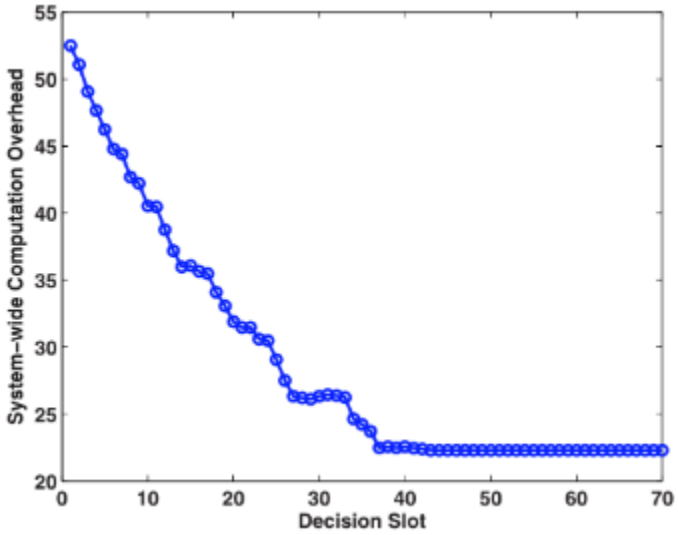


Figure 14 Dynamics of system-wide computation overhead

According to the figure at above, the reduced overhead of EMU offloading algorithm tends to be steady when the number of devices increase, which is same as the information shown in Figure 11.

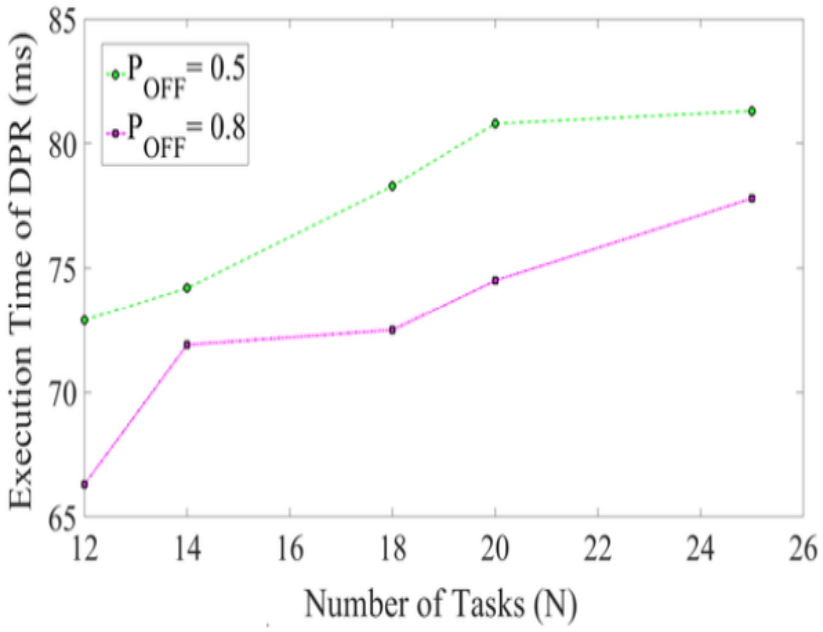


Figure 15 Execution Time of DPR for a different number of tasks

According to the figure at above, the execution time keeps increasing when the number of devices is not very big and tends to be steady when it becomes big enough, which is the same as the Figure 13.

7. Conclusions and Recommendations

7.1. Summary and Conclusions

MEC technology focuses on the end-user customer experience. The end device is defined as a mobile device. The user is not only able to access deployed services but also able to have local computing resource. Basically, on the local device, it has to deploy the client. The core tasks includes task dividing policy, distribution of the subtasks on local devices or task offloading on mobile edge devices.
Problems with high dynamic situation:

- Load variance of services
- Load variance of local devices and the changes of available resources including battery and cpu.
- Mobility that users' information has to be updated frequently.

Optimization goals:

- Application response time
- Battery consumption
- Cost

7.2. Recommendations for Future Studies

Since the MEC environment is complicated, many factors may impact the result. It is significant to do tradeoff decisions based on the actual needs. The challenges we may face in future work will be the topics like:

- Building multi-tenancy architecture by using pure cloud technology:
  MEC servers cannot rely on the advantages of high availability and high performance from large-scale data center since the location may be far away and the capability is lower. Therefore, it needs pure cloud technology to separate the functionalities of the services on different MEC servers to ensure each MEC server has high availability and high performance with limited resource for the specific service.
- Network control system of separating services of control level and user level:
  If MEC servers are deployed close to access point of network edge, it requests massive network configuration and maintenance. Network control system simplifies the network construction and prevents routing loop and high load issues.
- Real-time entertainment:
  High-definition video service based on MCDN, AR/VR mobile interactive gaming and intelligent vehicles request MEC servers have fast response to the real time tasks of calculation.

8. Bibliography

[1] X. Chen, L. Jiao, W. Li and X. Fu, "Efficient Multi-User Computation Offloading for Mobile-Edge Cloud Computing," in *IEEE/ACM Transactions on Networking*, vol. 24, no. 5, pp. 2795-2808, October 2016.
[2] H. Shahzad and T. H. Szymanski, "A Dynamic Programming Offloading Algorithm Using Biased Randomization," *2016 IEEE 9th International Conference on Cloud Computing (CLOUD)*, San Francisco, CA, 2016, pp. 960-965.

9. Appendices

9.1. Program flowchart

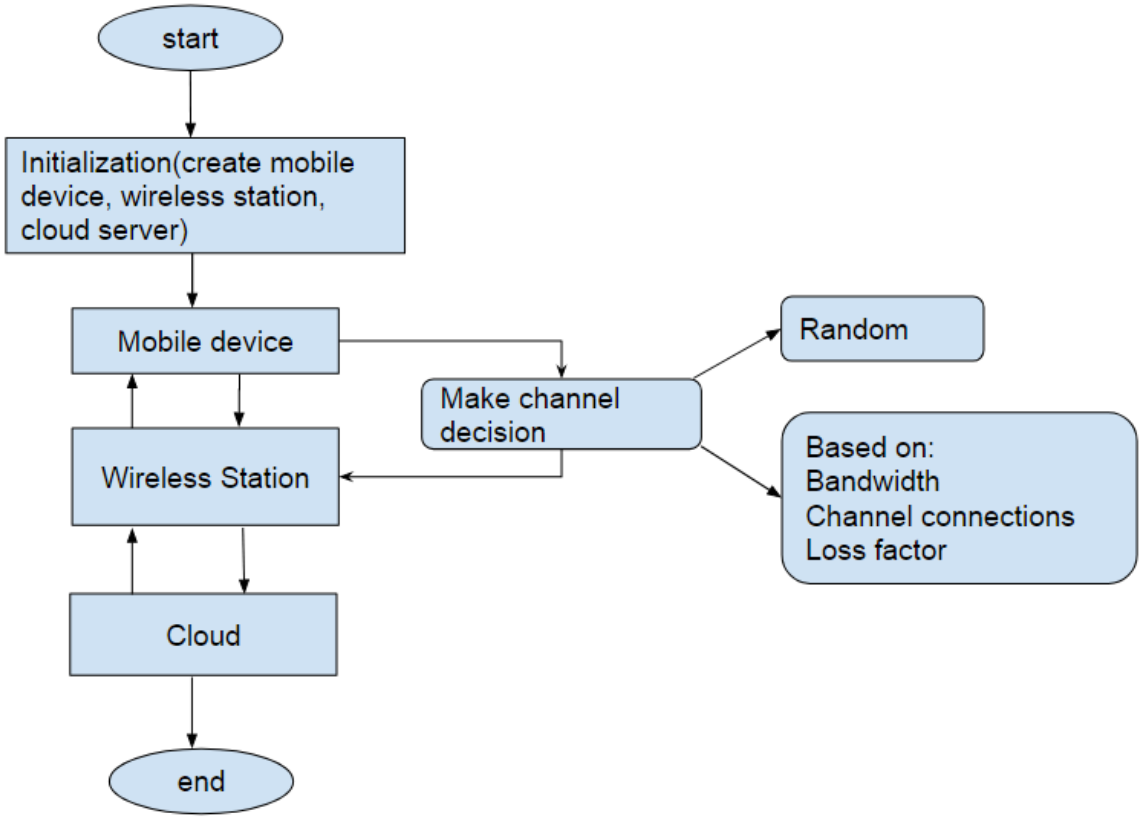


Figure 16 Program flowchart