Faculty of Natural and Mathematical Sciences
Department of Informatics

The Strand
Strand Campus
London WC2R 2LS
Telephone 020 7848 2145
Fax 020 7848 2851

7CCSMPRJ

Individual Project Submission 2016/17

Name: Govardhan Kolur

Student Number: 1559866

Degree Programme: M.Sc Computer Systems Engineering with Management

Project Title: Proactive Energy Trading in Small Cell wireless networks.

Supervisor: Dr. Reza Nakhai

Word count: 9249

✓ I agree to the release of my project

☐ I do not agree to the release of my project

Signature: Govardhan Kolur

Date: 21-08-2017

ABSTRACT

The objective of the project is to create a real time energy trading model of a Cloud based Radio Access Network. The project considers a Cloud based small cell network and aims to provide a cost efficient solution to the real time energy trading model by predicting the future energy needs of the model.

The project uses two algorithms, algorithm 1 for the calibrated forecasting and algorithm 2 for the MAB strategy based on the paper by Setareh Maghsudi, Slawomir Stanczak to obtain a prediction strategy for the future energy needs.

We aim to adapt the channel selection model in the aforementioned paper to the real time energy trading model. The aforementioned algorithms work together so as to provide us a forecast of the energy needs of the network so that the loss pertaining to the prediction strategy is minimized.

ACKNOWLEDGEMENT

I would like to thank my project supervisor Dr. Reza Nakhai and my PhD guide Xinruo Zhang for all their prompt insight and input without which the dissertation wouldn’t be possible. I would like to give my regards to my family and my close friends who helped me a lot during this project and for their moral support.

TABLE OF CONTENTS

LIST OF FIGURES

Chapter 1

INTRODUCTION

1.1 Project Overview

The project seeks to create an efficient system for the problem of energy trading for cloud based small cell wireless networks. The demand for telephone and internet services has been increasing for the period of the past decade and per the amounts of data that needs to be handled by the wireless network, the overall cost required to maintain the wireless network also has been increasing. The demand for data has increased from only telephone services being provided in the last decade to multimedia content on social media apps that has made the demand for sophisticated wireless networks even more significant.

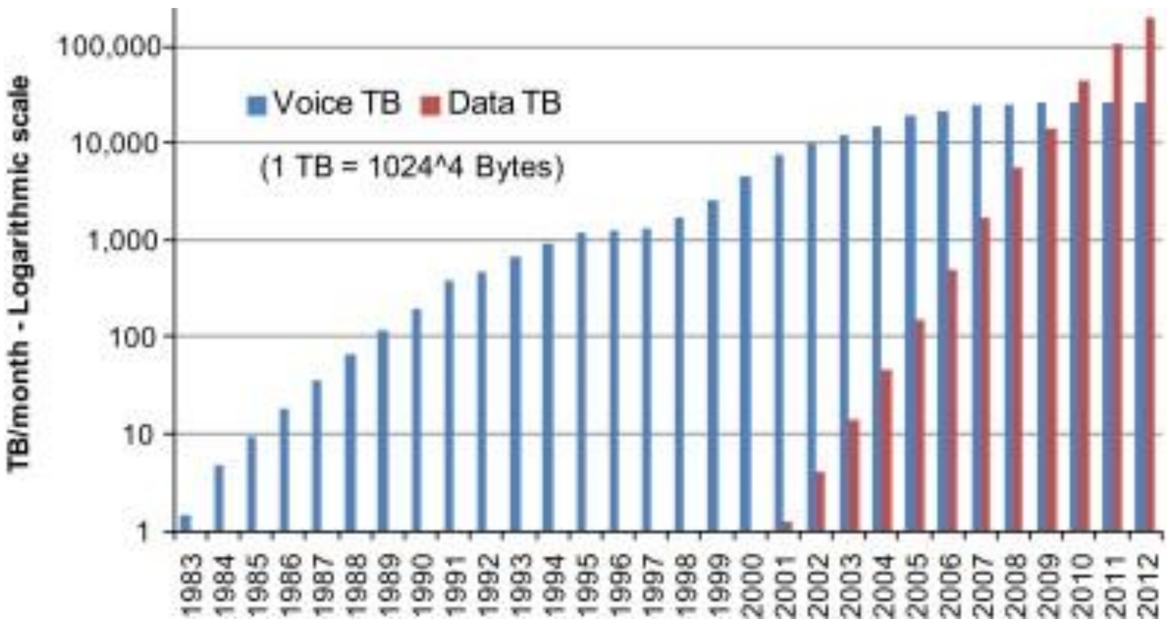

Fig 1: Demand for data in Terrabytes from the years 1983 – 2013

The wireless network gets its energy from both renewable energy sources and non-renewable energy sources. The current scenario is such that the wireless network operators are making a significant migration from non-renewable energy sources to renewable energy sources so that the penetration of renewables is larger than the penetration of non-renewable energy sources. This sort of migration raises a new challenge, that is, of the renewable energy sources being unreliable and harder to procure in real time due to the energy being generated as a ramp function as opposed to an impulse function when it comes to non-renewable energy sources. These differences are mainly due to the nature of the energy source.

Another challenge is, that the nature of wireless networks is such that it is susceptible to noise and attenuation from other devices transmitting through the channel. Thus, even as the distance between devices increases the noise and attenuation factor increases. A solution to this problem could be to increase the number of base stations but this only increases the interference between different base stations and increases costs in terms of the Total Cost of Ownership (TCO) of deploying the base station. Thus, to keep the costs at minimum it is not recommended to increase the number of base stations, especially keeping the QoS (Quality of Service) standards of wireless networks in mind.

We thus define the solution to be the fact that, if we can predict the energy needs of the wireless network in advance, then we can make a quote to the renewable energy source provider so that the overall Operating Expenditure (OPEX) of the system is reduced. We can thus use the renewable and non – renewable energy sources in such a way that the expenditures are minimum with respect to the demands of the wireless network.

We also note here that the Capital Expenditure (CAPEX) and Operational Expenditure (OPEX) reduces greatly by adopting cloud based small cell wireless networks as compared to traditional centralized wireless networks. We are trying to reduce the OPEX even more significantly through the proposed project.

Our proposed energy trading model will consist of a Calibrated Forecaster algorithm which is used to calibrate the bandit learning strategy, which is a type of machine learning strategy aimed at improving the overall cost of energy of the wireless network. The bandit learning strategy we choose to implement is the MAB (multi armed bandit) strategy. This strategy is discussed in detail in the forthcoming sections.

1.2 Project Objectives

The project aims to implement the proposed model using the algorithms used in the paper by Setareh Maghsudi, Slawomir Stanczak. In the algorithm, there are two base stations which are the players playing against each other. These players aim to maximize the reward of playing a particular strategy by observing the actions of its opponent. The player generates a stochastic curve of its prediction of which strategy the opponent will choose and adjusts its strategy accordingly. Henceforth, we can call the strategy as arms which are the energy packets that are chosen by the base station, this model is discussed ahead in full detail.

Once the stochastic curve has been generated, we next implement a MAB algorithm, which is a calibrated bandit strategy, for the player to choose the arm which maximizes the reward for playing the strategy. The reward we speak of is a function of the total power consumed by the base station and is a function of the SINR which in turn is a function of the interference between the base stations.

The wireless network model that we speak about can either be a case of orthogonal access in which case the interference between the base stations can be neglected or it can be non-orthogonal access wherein the interference depends on the total power consumed by both the base stations and the SINR.

1.3 Structure

Chapter 2 will elucidate the background information of the system, chapter 3 would illustrate the model of the systems and explain the whole process block by block. In chapter 4 we have the simulation results of the model. In Chapters 5 and 6 we conclude the report and we make suggestions for any further study based on the topic.

Chapter 2

BACKGROUND INFORMATION

2.1 Overview

In this chapter, the background information regarding the proposed model has been introduced. This information, along with the information from the succeeding chapters is used to construct the model.

2.2 Literature Review

In this paper, we explore various ideas set out in terms of calibration strategy, MAB strategy, optimization, projection of vectors calculation of rewards, to name a few. We have explored the ideas in various scientific papers and are thus able to implement these concepts to our study, which is mainly minimizing the energy cost of operating the network.

In the paper by Setareh Maghsudi, Ekram Hossain we explore the various learning strategies such as stateless, stateful bandits and CMAB learning strategies to implement the appropriate strategy for our purpose.

In the paper by Setareh Maghsudi, Sławomir Stanczak, we explore the idea of calibrated forecasting and how it gives us a strategy to predict the opponents behaviour and allows us to make a decision that will minimize the regret of choosing a particular strategy. The paper tells us that we need to form a stochastic distribution of choosing a particular strategy by using a distributed algorithmic solution. After the stochastic curve is obtained, we choose a strategy based on the information provided by the forecaster and the reward calculated using the environment variables and the MAB strategy.

In A Geometric Proof of Calibration by Shie Mannor, Gilles Stoltz we explore the calibration strategy in more detail.

In the paper by Yoav Freund, Robert E. Schapire, we explore the idea of multiplicative weights algorithm to calculate the update rule for the stochastic curve we obtain.

In the paper by Daniel Pérez Palomar we explore the idea of decomposing the primal problem into simpler sub-problems to solve a convex optimization problem.

Finally, in the paper by Wan Nur Suryani Firuz Wan Ariffin, Xinruo Zhang, Mohammad Reza Nakhai we explore the CRAN network environment so that we can calculate the reward of selecting each strategy based on its SINR.

2.3 Cloud Radio Access Networks (C-RAN)

Traditionally, cellular networks were designed such that there were only standalone base stations. Each standalone base station, or BTS only covered a small coverage area over a continuous region. Each BTS would receive and transmit its own signal to the mobile, and forwarded the data to and from the terminal onto the core network. Each base station used to be a separate, independent system with its own independent resources. Because of limited frequency spectrum the problem at hand was that there was limited frequencies available and the base stations reused the same frequency which caused a lot of interference between base stations.

There were more limitations to be considered with the traditional network, which was that that the base stations were hard to install and operate, adding to capital requirements for the same, the interference problem was the second problem, and the third problem was that even though the average utilization rate was pretty low of each base station, the processing resources could not be reallocated.

The solution to these problems was to remodify the existing cellular network with a central processor (CP) and many low cost Remote Radio Heads (RRHs). The advantage of this was that the cost which was incurred in deploying many base stations was reduced due to the low cost of a remote radio head. Deploying these RRHs in such a fashion also ensured that resource allocation, which was an earlier identified problem, could be taken care of in a centralized fashion.

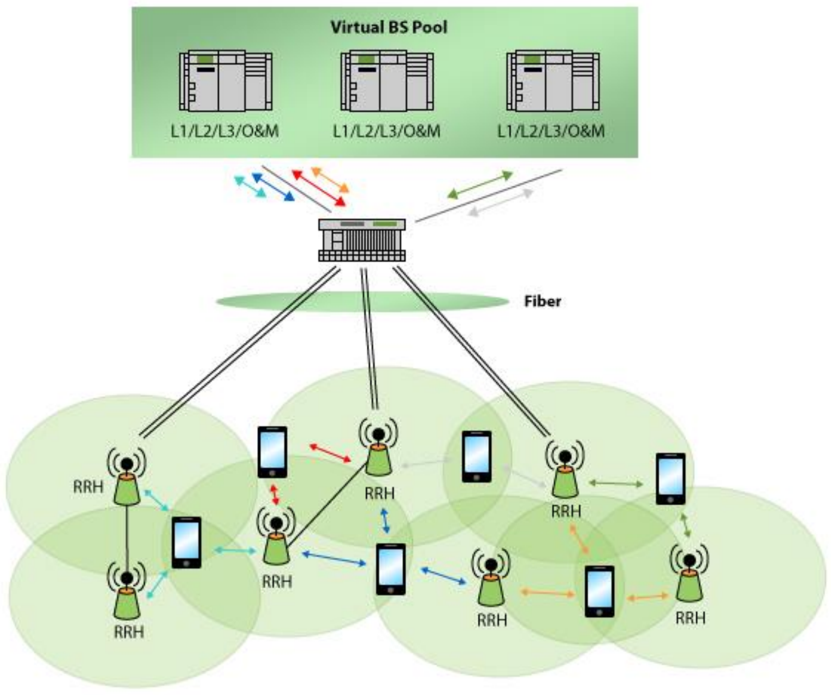

Figure 2: A cloud radio access network.

The interference problem also can be taken care of by allocating this network instead of a traditional network. Thus all problems that were identified can be solved.

2.4 Calibrated Forecaster

Consider a centralized strategy for choosing the aforementioned energy package from an energy spectrum, in this case, the base stations communicate directly with the power station to utilize the energy spectrum, thus their decisions are not monitored and governed by any entity such as a central controller. The base stations compete with each other to maximize the reward which in turn is dependent on the total transmission power and SINR which depends on the interference between multiple users of the base station. This information can be difficult to obtain from a cellular network. Also, we want to reduce the adverse effects of this strategy, mainly the interference between multiple users of the base stations.

We thus introduce an algorithm wherein we propose a distributed algorithmic approach to selection of packages from an energy spectrum wherein only vacant energy packets are used.

A calibrated forecasting strategy is described as follows, we first consider a stochastic process with finite states. These states are governed by an objective probability distribution that is yet unknown. The algorithm works so as to assign deterministic predictive probabilities to yet unknown states so that with time, the future probabilities, i.e, the probability that we forecast settle toward the probabilities that are unknown.

The forecaster can be said to be calibrated, if, the forecasted probability converges with an empirical stochastic distribution.

A theoretical approach to calibrated forecasting shows a diminishing regret of choosing the energy packets as compared to a global optimum solution that is obtained by a centralized learning strategy.

Probabilistic forecasts can thus be defined as variables that take the form of stochastic probability densities. The main aim of the algorithm we propose is to maximize the sharpness of the cumulative distribution obtained through an iterative algorithm and by adopting along with it a suitable learning strategy.

2.5 Multi Armed Bandit Strategy

The demand for internet and telephone services is increasing. In the future, wireless networks are expected to utilize new technologically advanced concepts in networking, communication and energy efficiency such as D2D communications, small cell networks and energy harvesting. The concept of centralized resource management needs to be tackled, which are excessively complex, thus we need to develop a distributed approach which can efficiently model the competition between various users of the wireless networks so that efficient allocation of resources can take place.

We thus introduce the MAB algorithm. Multi armed bandits is an optimization problem run in a sequential fashion, in which, there is an action set, over which the player plays a particular action in order to receive some reward. The reward is unknown to the player in advance, but on playing the action, the reward of the particular action or arm is revealed. The player may thus incur some loss or cost that is incurred by him not playing the best arm (the action with the highest reward), this loss is known as the regret. Thus, for each player in the MAB strategy, the objective of the game is to minimize the regret over an iterative algorithm, and maximize the regret accumulated.

The MAB strategy benefits from variations in the wireless network model such as different levels of information available to the base stations as well as different types of random variations.

2.6 Block diagram for full strategy

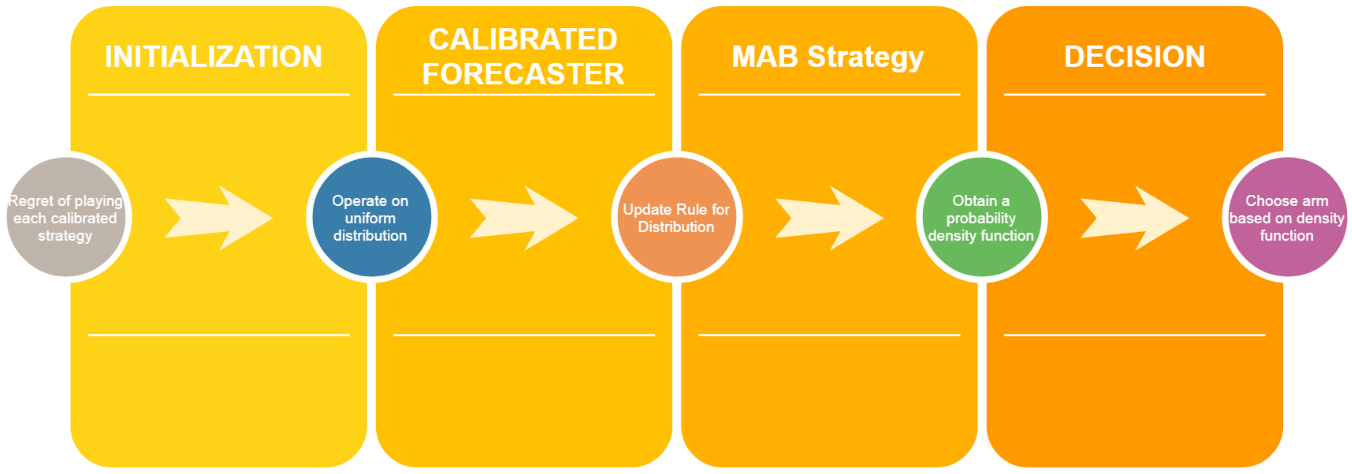
INITIALIZATION
CALIBRATED FORECASTER
MAB Strategy
DECISION
Regret of playing each calibrated strategy
Operate on uniform distribution
Update Rule for Distribution
Obtain a probability density function
Choose arm based on density function

Figure 3: Block Diagram for strategy

A block diagram for the full strategy has been given above. The model consists of initialization of parameters, which acts as the input to the calibrated forecaster. The MAB strategy then implements the strategy of the calibrated forecaster and a final decision is made on which strategy, i.e, arm to choose at the end of each period.

2.7 Adaptive Game Playing Algorithm (Multiplicative Weights Algorithm)

An algorithm that we use to design the update rule that is mentioned in the block diagram is to use the adaptive game playing algorithm which is also known as the multiplicative weights algorithm.

Consider a game played with two players, player 1 and 2 wherein each player incorporates a certain strategy. Let player 1's strategy be denoted using rows and player 2's strategy be denoted using columns. We can thus define the loss of playing a particular strategy for the row player as a certain quantity. The objective of the game, from player 1's perspective is to suffer a loss that is no lesser than that of the game.

For example, if we assume a game M, the first player chooses strategy i and the second player strategy j then we can say that the loss experienced by the player is M(i,j). The objective of player 1 would thus be to ensure that M(i,j) is lesser than the value of matrix M.

We could thus assume that we could ensure that this happens through a simple minmax theorem. But there are drawbacks to this theorem, namely:

1. M may be unknown thus we cannot calculate loss for an unknown matrix.
2. M may be excessively large and a minmax strategy and optimization could be infeasible.
3. The strategy of the column player may be non-adversarial and thus the algorithm could actually produce an output that is significantly lower than the game value, which is undesirable.

The algorithm that is proposed in later sections aims to solve this problem by using a algorithm running over finite rounds to obtain bounds that are non-asymptotic. Thus the problems identified with the minmax strategy could be eliminated.

2.8 Convex Primal Decomposition Algorithm

This particular algorithm proposes a primal decomposition approach to the optimization problem that we encounter while calculating the projection of one vector onto another vector in the Calibrated Forecaster design algorithm.

The aim of the algorithm is to decompose the convex optimization problem into in various sub-problems that are easier to solve such that a larger problem governs them. The paper by Daniel Pérez Palomar uses this algorithm to make the solution obtained through convex optimization from single channel model to multiple channel model. Since model is based on simulating energy packets as arms instead of channels, we can say that the convex primal decomposition algorithm pertains to multiple arms in our paper.

Chapter 3

System Design

3.1 Overview

As we had enunciated earlier, the main aim of this paper is to simulate our model pertaining to the problem identified and simulated in the paper by Setareh Maghsudi, Slawomir Stanczak. We however change the algorithm pertaining to the problem of channel selection in the paper to energy packets as arms of the MAB algorithm.

The model is implemented in the following fashion. First, we design the calibrated forecaster by initializing reward and regret based on a calibrated strategy, then we use this algorithm to solve the problem of arm selection in the MAB algorithm. The primal decomposition algorithm is used in the calibrated forecaster algorithm to get a solution to the optimization problem (i.e, to calculate the projection of a vector on another vector) and the adaptive game playing (multiplicative weight) algorithm is used as the update rule for the forecasted probabilities.

3.2 System Model Formulation

Figure 4 illustrates flow chart for the system model. The algorithm is based on the implementation of the flow chart.

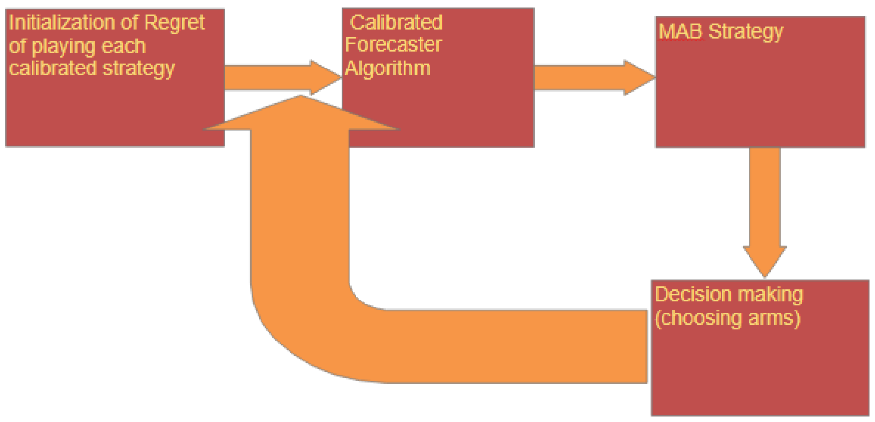
Initialization of Regret of playing each calibrated strategy
Calibrated Forecaster Algorithm
MAB Strategy
Decision making (choosing arms)

Figure 4: Flow Chart for strategy

3.2.1 Designing The Calibrated Forecaster

Consider an experiment with a set of outcomes that are finite and let the cardinality, (no. of outcomes of the experiment) be D, and let δ_{d_t} represent the dirac probability distribution of some outcome d over a finite set of outcomes *D* at time t. At time t, the forecaster returns an output probability distribution P_t over the set of outcomes. The forecaster is said to be calibrated if the following condition is satisfied:

$$\lim_{T\to\infty}\left\|\frac{1}{T}\sum_{t=1}^{T}1_{\{P_t=P<\epsilon\}}(P_t-\delta_{d_t})\right\| < \epsilon$$

Thus to design the forecaster we consider a player with the probability distribution P_t over a finite set of outcomes Q = {1,2,3,…….N_{ϵ_r}} wherein the action set denotes the possible strategies that can be played by the player such that the infinite set of outcomes that were presented earlier reduce to a finite set of N_{ϵ_r} outcomes. The condition of calibration also changes accordingly to reflect the finite set strategy.

According to the paper that we study, it an adaptive strategy consisting of multiple players, a strategy wherein there are selfish players and they learn an optimal solution based on multiple interactions with a random environment, and finally stop when they achieve an equilibrium.

We then propose an algorithm to implement the calibrated forecaster such that the condition of calibration is achieved. The algorithm is as follows:

Algorithm 1:

Step 1: Define a set of integers in increasing order T = 2^r for r = 1,2……., Tr is the number of time slots in the particular period.

Step 2: Define a game, wherein there are two players, the first player has the action set *I*= {1,2,3….,N_{ϵ_r}} and the second player has action set *D* with cardinality D.

Step 3: Define ϵ as $2^{-r(/D+1)}$ for each period.

Step 4: Define the regret in a vector for the 1st player as u(q,d) = (0,0……..p_q-δ_d } for each q ∈ {1,2,3…..,N_{ϵ_r}} and for each d ∈ D.

Step 5:

- Write a vector with (DN_{ϵ_r}) dimensions of $\mathbb{R}^{DN_{\epsilon r}}$ as vectors of N_{ϵ_r}dimensions having sub-components in $\mathbb{R}^D$ i.e, X = {X_1,…….,$X_{N_{\epsilon_r}}$} , $X_l \in \mathbb{R}^D$, ∀ l = {1,……., N_{ϵ_r}}.
- A ball of radius r has a subset *F* of radius ϵ_r around (0,… p_q-δ_d ,…,0) for the l2 norm for which the F is a closed convex set

Step 6: Regret upto time T = 1≤T≤T_r is defined as follows:

$U_T = \sum_{t=1}^{T} u(Qt, dt)$

```
Step 7: repeat the following steps
Step 8: for T = 1:$T_r$
Step 9:         if (r = 1 && t = 1)
Step 10:        An action Q is picked from a uniform distribution ψ = {$1/N_{\epsilon_r}$,....., $1/N_{\epsilon_r}$}
                where ψ is the output probability distribution
Step 11:        else if (r>1 & t =1)
Step 12:        An action Q is picked from a neighbourhood of the output probability
                distribution.
Step 13:        else
Step 14:        An action $Q_t$ is selected from *I* from the output probability distribution after
                designing the update rule such that the following condition is satisfied:
                $(u_{t-1} - \prod_F (u_{t-1})).(u(\psi_t,d) - \prod_F (u_{t-1})) \leq 0$
Step 15:        end the if statement
Step 16:        end the for loop
Step 17:        Calculate the regret of the current period as the regret of the last time slot.
Step 18:        If $u^r > \epsilon_r$ then
Step 19:        Set r to 1 and t to 1
Step 20: else
Step 21:        Set r to r+1 and t to 1
Step 22: end if statement
Step 23: run until convergence is obtained (r should be large enough such that $\epsilon_r \approx 0$.
```

3.2.2 Designing MAB strategy

The MAB strategy is also called the bandit strategy. In the reference paper that we are trying to model, the problem regarding channel selection is modelled as using the bandit learning strategy. We use the algorithm to model the energy packet selection problem that we identified in the earlier sections.

In the strategy that we implement, the game is played in periods r = 1,2,3…. of incremental length T_r'. A vector Z_r is defined for the same time period so that the vectors satisfy the following conditions:

We assume that T_r and Z_r are selected such that

a) $[T_r'.Z_r]$ is in increasing sequence.
b) $\lim_{R\to\infty} \sum_{r-1}^{R}[T_r'Z_r] \to \infty$
c) $\lim_{R\to\infty} \sum_{r-1}^{R}[T_r'Z_r] / \sum_{r-1}^{R} T_r' \to 0$

At each period r, we select $[T_r'Z_r]$ trials as exploration trials and the rest are used as exploitation trials.

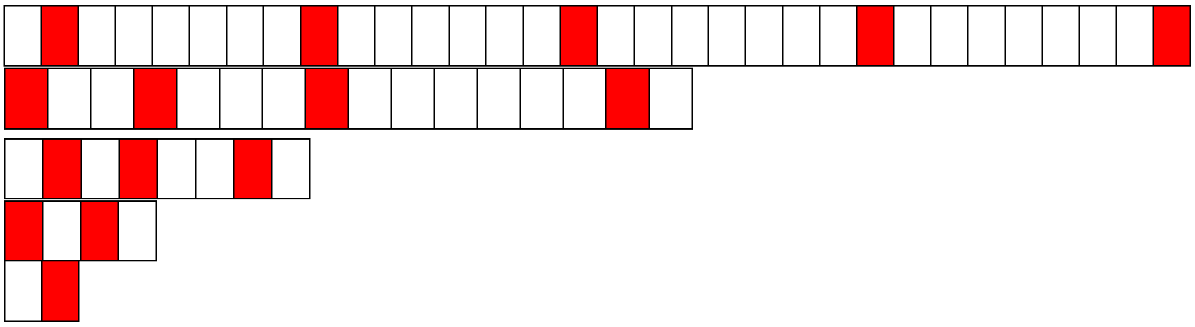
Figure 5: Exploration vs Exploitation trails, red trials denote exploration, each row is a period.

Algorithm 2:

```
Step 1: Define a set of integers in increasing order such that Tr = $2^r$ for r in increasing order
such as r = 1,2,3…….

Step 2: Define a set of numbers in decreasing order Zr = r/$2^r$ for each period in time.

Step 3: Set the index of the current period r = 1 and an exploration parameter $\gamma<<1$.

Step 4: Repeat the following steps,

Step 5: Select r exploration trials from the first and last trial index [1+$\sum_r T'_{r-1}$, $\sum_r T'_r$]
randomly and uniformly.

Step 6: Declare a for loop such that   t = s + $\sum_r T'_{r-1}$, such that $1 \leq s \leq T_r$

Step 7: If the current trial is for exploration

Step 8: Choose an arm at random with probability $\gamma$

            With 1- $\gamma$ probability

                    Run the calibrated forecaster algorithm. Calculate the mean reward of each
                    arm and choose arm with the highest reward

Step 9: else

Step 10 & 11:  Run the calibrated forecaster algorithm and calculate the mean reward and
               choose the arm with the highest reward function.

Step 12: end the if statement

Step 13: The arm is chosen and the reward is noted

Step 14: Actions of the opponent are observed.
```

Step 15: The reward estimate is improved.

Step 16: end the for loop.

Step 17: Increment r

Step 18: Run the program until convergence.

3.2.3 Designing Adaptive Game playing algorithm (multiplicative weight)

The adaptive game playing algorithm is used for the update rule to update the output probability distribution in the calibrated forecaster algorithm.

A two player game in normal form is considered. The definition of the game is through a matrix with m,n dimensions M such that it has m rows and n columns. If player 1 chooses strategy i. and the player 2 chooses strategy j, the loss can be referred to as M(i,j).

Two types of strategies come into picture, one is pure strategy and the other is mixed strategy. The mixed strategy is the loss over all the rows or columns whereas the pure strategy is associated only with a particular row (for example i) chosen by the player. The mixed strategy of the first player is denoted by P and the mixed strategy of the second player is denoted by Q. When two mixed strategies are used, we can say the loss experienced is M(P,Q) and when a mixed strategy and a particular strategy (for example i) is used then we can say the loss experienced is M(i,Q) or M(P,j).

The algorithm aims to select mixed strategies for the player and the environment adaptive strategy that is played over various rounds. Normally, we use this algorithm for the row player and denote the second player to be the environment. We thus play this algorithm over rounds t = 1…….T

The proposed algorithm is as follows:

Algorithm 3

Step 1: Player 1, who is learning, chooses a mixed strategy, say P_t

Step 2: The opponent, or the environment, chooses a mixed strategy, say Q_t (The action may be played with the knowledge of the action of the other player)

Step 3: Player 1 calculates how much loss $M(i,Q_t)$ for each pure strategy i.

Step 4: The player experiences loss factor of $M(P_t,Q_t)$

This algorithm is played repeatedly over many rounds. And thus we design the update rule.

The update rule is designed as follows, the learning strategy gives us a probability distribution P(1) for the initial round. We update the distribution over an iterative set of rounds like this:

$$P_{t+1}(i) = \frac{P_t(i).\beta^{M(i,Qt)}}{Z_t}$$

Where

$$Z_t = \sum_{i=1}^{n} P_t(i).\,\beta^{M(i,Qt)}$$

By playing this adaptive game playing algorithm, we can calculate the regret or the loss of playing each calibrated strategy and thus design the update rule for the probability distributions.

3.2.4 Designing Convex Primal decomposition algorithm

We use the convex primal decomposition algorithm to calculate the optimization problem that is defined later in the algorithm, to, calculate the projection of a vector on another vector. By calculating the projection, we define a loss function or a regret function using the aforementioned multiplicative weight algorithm. We thus aim to produce a loss function for the same and then calculate our output probability based on this.

For the purposes of this problem, we consider the projection of a vector with N dimensions x_0 on a simplex, a vector x such that all the elements of x_0 are made positive. A convex problem is defined and solved using an iterative algorithm.

We can define the problem as such

Minimize $||x - x_0||^2$

Such that $x_i \geq 0$ for $1 \leq i \leq N$

$\sum_{i=1}^{N} x_i \leq \varepsilon$

The optimal solution is thus,

$x_i = (x_i - \mu)^+$

An iterative algorithm is proposed to solve the aforementioned problem,

---

Algorithm 4:

Input: The vector that you want to calculate the projection of (x_0) and constraining point ε

Output: The vector containing the projection *x*

Step 1: First try μ = 0: if $\sum_i (x_i)^+ \leq \varepsilon$ then obtain $x_i = (x_i)^+$, finish the algorithm.

Step 2: Derive a μ>0 such that $\sum_i x_i = \varepsilon$ thru the following steps:

Step 2.0: Reorder the vector in decreasing order, set Ñ = N and set $x_{\tilde{N}+1} = -\infty$

Step 2.1: Check if $x_{\tilde{N}} > x_{\tilde{N}+1}$ and $x_{\tilde{N}} > \sum_{i=1}^{N}(x_i - \varepsilon)/\tilde{N}$ then the condition is true, now go to step 2.2 else reject the condition and check for the exit condition agai by setting Ñ = N-1 and go to step 2.1

Step 2.2: Define $\mu = \sum_{i=1}^{N}(x_i - \varepsilon)/\tilde{N}$, calculate the optimum as defined before, $x_i = (x_i - \mu)^+$ and finish.

Once we calculate the optimal solution we can say that the projection of one vector over the other has been calculated and we can use this for further computation of the loss. The proof for this theorem has been proven in the paper by Yoav Freun, Robert E. Schapire.

3.2.5 Working of the entire program

The entire program can be run as follows:

Working of the entire program

Step 1: Initialize the probability space of player 1 P as the following vector

P = {(0,1),(1/20,1-1/20)………………………………………………(1,0)},

These are probabilities for the player 1 to make a decision regarding arms/packets as calibrated according to 20 different strategies in the calibrated forecaster algorithm.

Step 2: Initialize the probability space of the player 2, δ as the following,

δ_d = {(0,1),(1,0)}

According to whether arm 1 is chosen or if arm 2 is chosen accordingly.

Step 3: Calculate the vector valued regrets as in the forecaster algorithm

Step 4: Calculate the vector valued regrets in each time slot

Step 5: Calculate the projection vector according to the primal decomposition algorithm.

Step 6: Use multiplicative weights algorithm for update rule for the forecaster probabilities.

Step 7: Choose a calibrated strategy according to distribution and time slot.

Step 8: Give this info to the MAB algorithm.

Step 10: Calculate the reward based on the information and choose an arm.

Step 11: This arm is thus chosen.

3.2.6 Designing Reward Calculation Algorithm

As mentioned before, in the MAB algorithm, we need to choose the arm that exhibits the highest reward after a certain set of calculations. Choosing the arm with the highest reward means that you are not choosing the arm that has the highest magnitude in each period, but rather you are calculating the reward for the arm and trying to optimize the overall energy cost of the system by trying to minimize its power consumption.

We thus, calculate the SINR of choosing each channel which depends on the power consumption of the whole network as such. We can thus ensure that the final power consumption is optimized while also considering the fact that the arm with the highest magnitude is not necessarily the arm which exhibits highest reward, thus we are not looking for maximum power but minimum energy cost which depends on minimizing the total energy consumed by each unit.

Keeping this in mind, we design the MAB algorithm to have the same characteristics of the nature of non-orthogonal access exhibited in the channel selection problem in the paper

Channel Selection for Network Assisted D2D Communication via no-regret bandit learning with Calibrated Forecasting by Setareh Maghsudi and Slawomir Stanczak.

The below formula helps us to calculate the reward based on the energy packet chosen. Its particulars are:

1. P_{ij} : The arm chosen for user I and BS j

   Notice that in this step we assume that there are 6 users using the wireless network with 3 base stations.

2. h_{ij}: The channel between the ith user and the j-th base stations.
3. K = the total number of arms in the j-th BS.

The formula for calculating the SINR is given below

$$\text{SINR}_{ij} = \frac{desired\ signal}{intra-cell\ interference + inter-cell\ interference + noise}$$

Thus,

$$\text{SINR}_{ij} = \frac{P_{ij}||h_{ij}||^2}{\sum_{k\neq \text{i}}^{K} P_{kj}||h_{ij}||^2 + \sum_{q\neq \text{j}}^{N}\sum_{m=1}^{K} P_{mq}||h_{iq}||^2 + N_0}$$

Once the reward has been calculated, we can say that the arm with the highest reward will be chosen and thus we can say that we are aiming to maximize the sinr which is a function of the overall power consumption.

Chapter 4

Simulation Results

We run the program through mainly the two algorithms, one the calibrated bandit strategy and the other the MAB strategy that uses the aforementioned algorithm. The simulation results have been discussed in the forthcoming chapter.

4.1 Simulation Model

For the purposes of simulation, we assume two energy packets, one 100 mW and the other 500 mW,

For the wireless network, we first construct a network with 2 RRHs and we consider that one RRH has exactly one user. We can thus assume that cell for RRH1 corresponds to only user1 and RRH2 corresponds to only user 2. We generate the reward using ZMCSCG random variables

We then construct the calibrated forecaster and give the input of the forecaster to the MAB strategy, wherein the MAB strategy picks an arm at every time slot based on the information of the reward corresponding to each arm. We are thus picking the arm with the highest SINR which is corresponds to the reward.

For the purposes of calculation of the reward we take the weighted average. The way the average is calculated uses the output of the forecaster, considers the average of the forecast and tries to pick the arm which is equal or almost equal to the average. Then we pick the probability of each arm being chosen that corresponds to the action picked.

Thus, we get two probabilities,

P1 corresponds to Q/number of quantized strategies and

P2 = 1-(Q/number of strategies)

We then calculate the reward based on whichever strategy player 2 assumes player 1 to pick. For probability P1 it assumes that arm 1 is going to be picked thus it says we calculate reward of choosing either arm based on that. Let's call them Reward1 and Reward2. We then calculate for probability P2 reward3 and reward4.

Thus P1*reward1+P2*reward3 gives the weighted average reward of the player 1 choosing the first strategy i.e, arm 1 and P1*reward2 + P2*reward4 corresponds to the player 1 choosing the second strategy i.e, arm 2.

We then calculate the time averaged reward so that we can check fo convergence of the reward. A reward converging in a trend shows that we can obtain the game finally settles at a point where both the players have chosen a particular arm based on the highest estimated mean reward.

4.2 Calibration Curve Simulation Results

For the purposes of plotting the reward curve and obtaining a good calibration curve, we want to compare the curves obtained at various time slots. Generally speaking, we need to obtain a more well defined curve with the edges of the curve tending almost toward zero and the maximum. Then we can say that we have obtained a well-defined curve. Thus this has been plotted in this section.

Result with 3 periods:

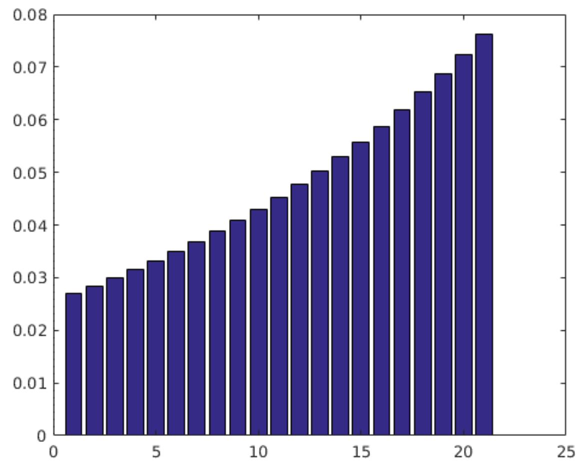

Figure 6: Calibration curve obtained with 3 periods

Result with 4 periods:

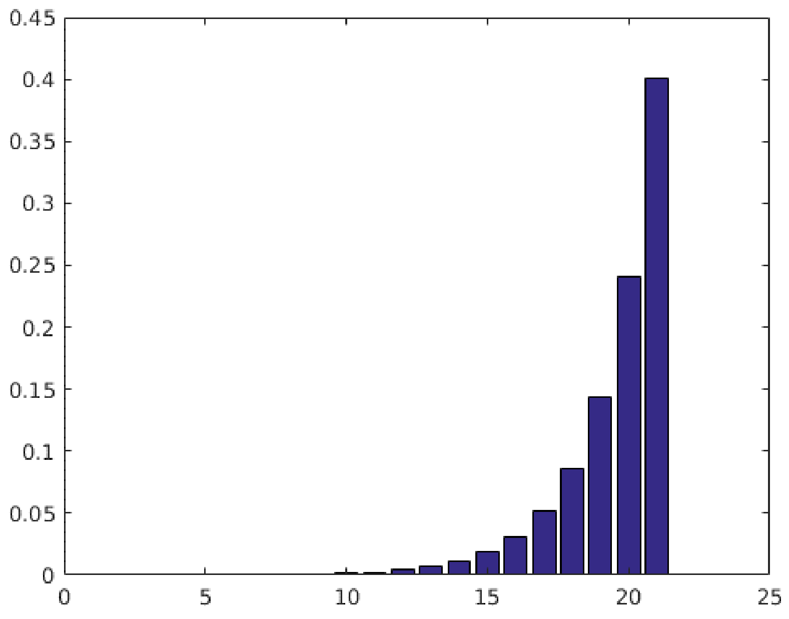

Figure 7: Calibration curve obtained with 4 periods

Result with 7 periods:

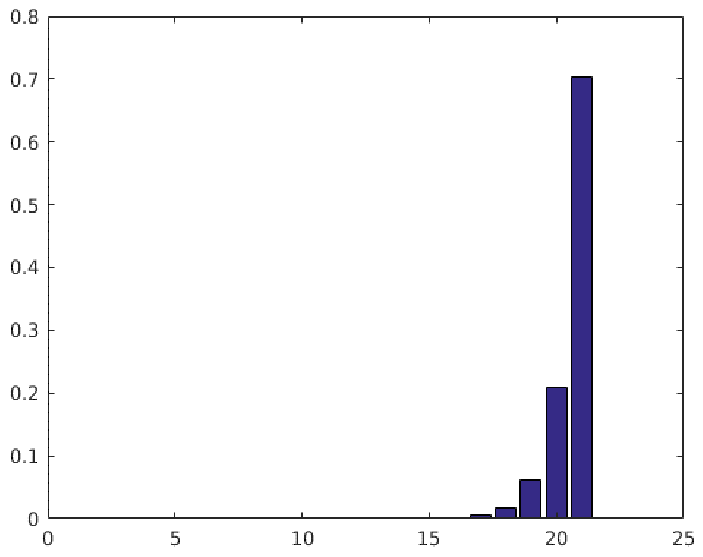

Figure 8: Calibration curve obtained with 7 periods

This curve has the quantized strategies as the X axis and the corresponding selection probability as the Y axis. We observe that as we increase the number of iterations, we get a more well defined curve. This corresponds to the fact that as we increase the number of iterations, we get the same arm chosen most of the time through the entire strategy being played in multiple periods.

4.3 Reward Simulation Results

Similarly, we plot the reward through the MAB strategy over different multiple periods of time. The aim of the reward function is to choose the arm that corresponds to the maximum SINR. The reward should converge to a particular value when played over the time periods and the convergence pattern should increase as we increase the number of periods.

Thus, by comparing the results over multiple iterations we can say if convergence is being achieved. This has been done in this section.

Reward for 3 periods:

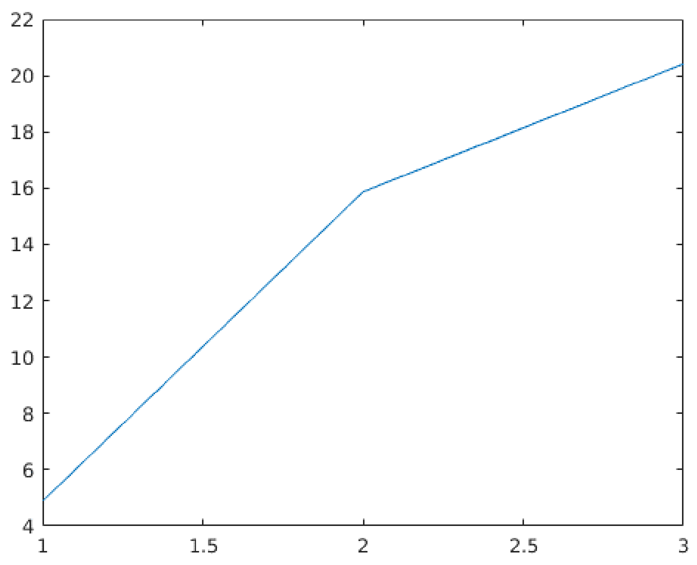

Figure 9: Reward curve with 3 periods

Reward for 6 periods:

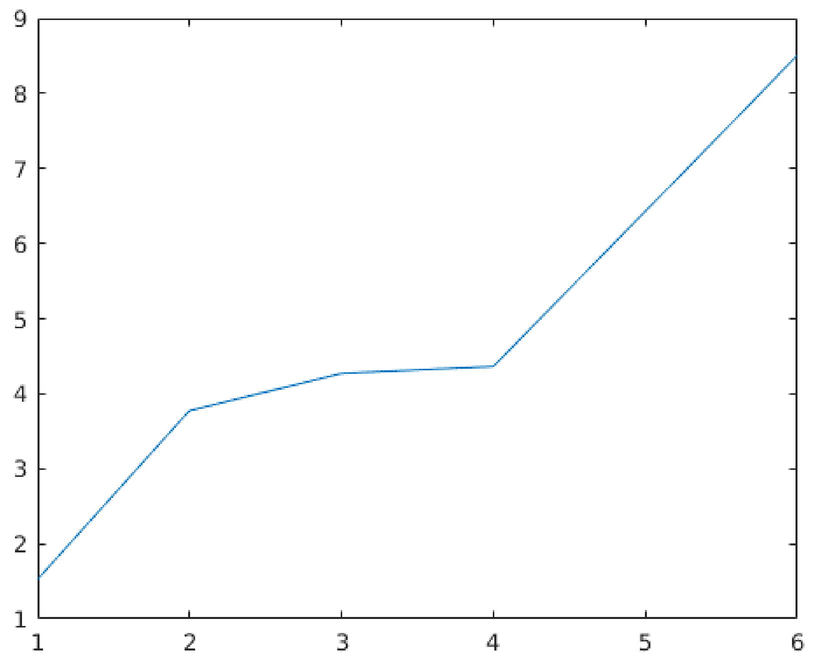

Figure 10: Reward Curve with 6 periods

Reward for 8 periods:

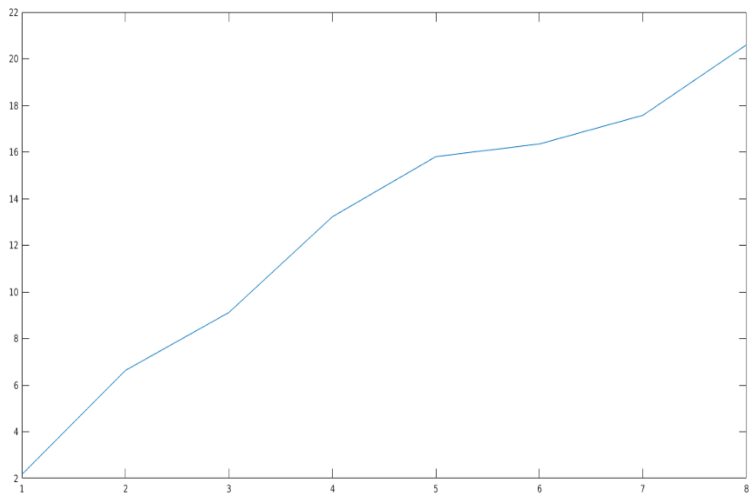

Figure 11: Reward curve with 8 periods

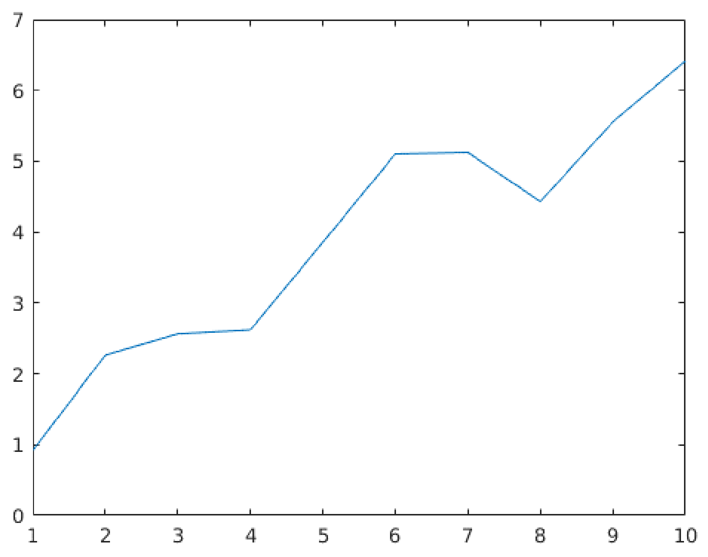

Figure 12: Reward curve with 10 periods

We observe that as we increase the number of periods, we get a convergence pattern, i.e, the points corresponding to the last few points in the reward curve in the plot are almost the same in successive iterations.

The implication of this convergence pattern being observed is that we can say that the same arm corresponding to the maximum SINR is selected when we run the algorithm over different periods of increasing length. This maximum SINR is yet again a function of the overall energy consumption.

Thus we can say that we choose the arm that maximizes the SINR which minimizes the energy consumption.

Chapter 5

Conclusion

In this paper, we try to apply the channel selection model proposed in our main research paper Channel Selection for Network Assisted D2D Communication via no-regret bandit learning with Calibrated Forecasting by Setareh Maghsudi and Slawomir Stanczak to the energy cost minimization model to actively predict the future energy needs of a wireless network. We did this by simulating the wireless network and then used a calibrated bandit learning strategy which is the calibration forecaster and the MAB learning strategy. We then used this strategy to predict the reward associated with each arm. The SINR function was implemented as a reward function, wherein the SINR corresponded to being more when the energy consumption was lesser.

The calibration strategy implements a probabilistic forecast of the energy needs of the base station by making a stochastic prediction of the which arm the opponent would choose. By doing this, we can say that the player or the base station has a probabilistic, statistical idea of what the other player is going to do in advance based on the regret which is accumulated over each time slot.

The fact that more energy did not correspond to more SINR was proven by the fact that the base station does not necessarily choose the arm with the highest magnitude but chooses the arm with the maximum SINR. Thus we can say that each base station is not looking to use the energy packet with the highest magnitude and thus use up more power but actually minimize the energy cost by picking the arm corresponding to the highest reward.

Thus we can say that the model aims to minimize the energy cost that is incurred to the base station by minimizing the power consumption.

References

1. Adaptive game playing using multiplicative weights by Yoav Freun, Robert E. Schapire
2. Convex Primal Decomposition for Multicarrier Linear MIMO Transceivers by Daniel Pérez Palomar.
3. Channel Selection for Network-Assisted D2D Communication via No-Regret Bandit Learning With Calibrated Forecasting by Setareh Maghsudi and Sławomir Stanczak
4. Multi-Armed Bandits with Application to 5G Small Cells by Setareh Maghsudi and Ekram Hossain.
5. A Decision-Theoretic Generalization of On-Line Learning and an Application to Boosting by Yoav Freund and Robert E. Schapire
6. A Geometric Proof of Calibration by Shie Mannor, Gilles Stoltz
7. Game Theory, On-line Prediction and Boosting by Yoav Freund Robert E. Schapire
8. Combinatorial Multi – Armed Bandit algorithms for Real – time energy trading in Green C-RAN.
9. Expanding mobile wireless capacity: The challenges presented by technology and economics by Richard N. Clarke
10. Heterogeneous networks, http://people.sutd.edu.sg/~tonyquek/?page_id=26 by Tony S, Quek.
11. Calibration, http://www.cs.cornell.edu/courses/cs683/2007sp/lecnotes/week5.pdf
12. Probabilistic forecasts, calibration and sharpness by Tilmann Gneiting, Fadoua Balabdaoui, Adrian E. Raftery

Appendix

This section consists of the source code that was used to simulate the environment and to get the simulation results. The appendix entails the two programs used to construct the calibrated bandit selection strategy and the code snippet to calculate the reward. A brief of the working of each program has been given.

First program:

exprunning2.m

This program consists of the MAB strategy that is used in the program. It consists of the selection of exploration and exploitation stages and choosing the arm in each stage according to exploration parameter.

caliprog.m

This program consists of designing the calibration forecaster the algorithm of which was discussed above. This program acts as input to the exprunning.m program. This program calculates the regret of choosing from 20 quantized strategies and giving input to the MAB algorithm. A rudimentary algorithm was developed in July by me and then it was improved upon subsequently.

Reward Calculation Code Snippet

This snippet calculates the environment variables for the C-RAN network and calculates the reward associated with using each arm by utilizing the channel specific information and intra and intercell interference by using ZMCSCG random variables.

projection.m

This program calculates the projection of a vector on another vector that is used for designing the update rule that is updated according to the loss vector. The loss vector is calculated using the projection.

exprunning1.m

```
%--------------------------------------------------
%for the MAB strategy, exploration and exploitation
%--------------------------------------------------
global vec_val_regT1  ; %initialize the vector valued regret
till time T
global vec_val_regT2 ;
global vec_val_regR1 ;
global vec_val_regR2;

R = [1,2,3,4,5,6,7] ; %initialize periods
Tr = 2.^R ; %number of samples in each period
T = Tr' ;
Tm = 1 ;

    t1 = 1 ;

    t11 = 1 ;
d = 0.3;
r=1 ;
G = 0 ;
reward_vector = zeros(1,length(R)-1) ; %initialize the reward
vector
reward_vector_2 = zeros(1,length(R)-1) ;



    channel_choosen2 = 1 ; %initialize channels choosen


    channel_choosen1_2 = 1 ;
for j = 1:21
actions_of_all_players(j) = j ; %joint action profile across
calibrated strategies
end
channel_choosen_vector = zeros(1,length(R)-1) ;
reward = zeros(1,2) ;
avg_reward = size(length(R)) ;
while (r<=7) %while the program is lesser than maximum number
of periods
  X = 0 ;
  S = 0 ;
  %F1_vector = zeros(r) ;
  Z = r./(2.^r) ;

    for j=1:r
        if (j>1)
        S = S+ T(j-1);
        X = X + T(j) ;
        else
        X = X+T(j) ;
        end
    end
    %channel_choosen = zeros(1,X) ;
    resultant = zeros(1,X-S) ; %to store exploration trials
    for j = 1:X-S
    resultant(1,j) = S+j ;
    end
    B = resultant(randperm(prod(size(resultant)))) ; %random
permutation of all the exploration trials choosen
    A = zeros(1,r) ;
    for j = 1:r
     A(j) = B(j) ;
    end
    A = A
    t = 0 ;
    G = G + 2^r ;

    for t = S+1:X %for the first trial in the period to the
last
     if (t == S+1)
         Tm = 1 ;
     end
        ans = t
                    for k = 1:r

                         if t== A(k)

                            flag = 1 ; %if it is an
exploration trial
                            break ;
                         end
                    end



  if (flag == 1)
      x = rand ;
      if x < 1-d %with probability 1-d
         channel_choosen1_2  = randsample([1,2],1) ; %choose a
channel at random
         channel_choosen2 = randsample([1,2],1) ;
%          prediction_forecaster = zeros(1,21) ;
%          for j = 1:21
%          prediction_forecaster(j) = 1/20 ;
%          end
      else %else with probability d


         %flag2 = 0 ;

%       for i = 1:3
%           reward(i) = log(reward(i)) ;
%       end
        channel_choosen2 = t11 ;
        reward1 = reward_calculation(1) ; %calculate SINR
reward
     reward2 = reward_calculation(2) ;
     prediction_forecaster1 = caliprog(channel_choosen2,r,Tm)
; %output the prediction of the forecaster
          U = mean(prediction_forecaster1) ;
             X1_1 = (prediction_forecaster1<=U) ;
            for i = 2:21
                if(X1_1(i) == 0)
                    action = i+1 ;
                     Q = action ;
                     break ;
                     else
                    action =
randsample(actions_of_all_players,1) ; %action is random
sample based on average of prediction curve
                    Q = action ; %to find index and
corresponding probability
                end
            end
             O11 = [Q/20] ;
                P11 = [1-Q/20]  ;
     reward_vector(r) = mean([reward1(1).*O11
reward2(1).*P11]) ; %calculate weighted average reward
      reward_vector2(r) = mean([reward1(2).*O11
reward2(2).*P11]) ;

     if(reward_vector(r) > reward_vector2(r))
         channel_choosen1_2 = 1 ; %choose channel with highest
reward
         avg_reward(r) = reward_vector(r) ;
         t1 = 1;
     else
         channel_choosen1_2 = 2 ;
         avg_reward(r) = reward_vector2(r) ;
         t1 = 2 ;
     end
           figure(2) , bar(prediction_forecaster1) ;

     reward_vector_log= 10*log10(avg_reward*20/100) ;
%calculate log vector (SINR in dB)
     cumulative_reward = cumsum(reward_vector_log)/r ;
%calculate cumulative reward
     figure(4), plot(reward_vector_log) ;
     figure(6), plot(cumulative_reward) ;
     channel_choosen1_2 = t11 ;
      prediction_forecaster2 =
caliprog(channel_choosen1_2,r,Tm) ; %output the prediction

           U = mean(prediction_forecaster2) ; %refer previous
section
             X1_1 = (prediction_forecaster2<=U) ;
            for i = 2:21
                if(X1_1(i) == 0)
                    action = i+1 ;
                     Q = action ;
                     break ;

                else
```

```
                    action = 
randsample(actions_of_all_players,1) ;
                    Q = action ;
                end
            end
             O12 = [Q/20] ;
                P12 = [1-Q/20]  ;
     reward_vector(r) = mean([reward1(1).*O12 
reward2(1).*P12]) ;
     reward_vector2(r) = mean([reward1(2).*O12 
reward2(2).*P12]) ;

     if(reward_vector(r) > reward_vector2(r))
         channel_choosen2 = 1 ;
         t11 = 1 ;
         avg_reward(r) = reward_vector(r) ;
     else
         channel_choosen2 = 2 ;
         t11 = 1 ;
         avg_reward(r) = reward_vector2(r) ;
     end
           figure(3) , bar(prediction_forecaster2) ;

     reward_vector_log= 10*log10(avg_reward*20/100) ;
     cumulative_reward = cumsum(reward_vector_log)/r ;
     figure(5), plot(reward_vector_log) ;
     figure(7), plot(cumulative_reward) ;

      end
  else

      reward1 = reward_calculation(1) ; %refer previous 
section
     reward2 = reward_calculation(2) ;
      channel_choosen2 = t11 ;
      prediction_forecaster1 = caliprog(channel_choosen2,r,Tm)
;
      U = mean(prediction_forecaster1) ;
             X1_1 = (prediction_forecaster1<=U) ;
            for i = 2:21
                if(X1_1(i) == 0)
                    action = i+1 ;
                     Q = action ;
                     break ;

                else
                    action = 
randsample(actions_of_all_players,1) ;
                    Q = action ;
                end
            end
             O11 = [Q/20] ;
                P11 = [1-Q/20]  ;
     reward_vector(r) = mean([reward1(1).*O11 
reward2(1).*P11]) ;
      reward_vector2(r) = mean([reward1(2).*O11 
reward2(2).*P11]) ;
```

```
     if(reward_vector(r) > reward_vector2(r))
         channel_choosen1_2 = 1 ;
         t1 = 1 ;
         avg_reward(r) = reward_vector(r) ;
     else
         channel_choosen1_2 = 2 ;
         t1 = 2 ;
         avg_reward(r) = reward_vector2(r) ;
     end
           figure(2) , bar(prediction_forecaster1) ;

     reward_vector_log= 10*log10(avg_reward*20/100) ;
     cumulative_reward = cumsum(reward_vector_log)/r ;
     figure(4), plot(reward_vector_log) ;
     figure(6) , plot(cumulative_reward);
         %flag2 = 0 ;

%      for i = 1:3
%          reward(i) = log(reward(i)) ;
%      end




         %flag2 = 0 ;

%      for i = 1:3
%          reward(i) = log(reward(i)) ;
%      end
channel_choosen1_2 = t11 ;
    [prediction_forecaster2]=
caliprog(channel_choosen1_2,r,Tm) ; %refer previous section
  U = mean(prediction_forecaster2) ;
            X1_1 = (prediction_forecaster2<=U) ;
           for i = 2:21
               if(X1_1(i) == 0)
                   action = i+1 ;
                    Q = action ;
                    break ;
               else
                   action =
randsample(actions_of_all_players,1) ;
                   Q = action ;
               end
           end
            O12 = [Q/20] ;
               P12 = [1-Q/20]  ;
     reward_vector(r) = mean([reward1(1).*O12
reward2(1).*P12]) ;
     reward_vector2(r) = mean([reward1(2).*O12
reward2(2).*P12]) ;

     if(reward_vector(r) > reward_vector2(r))
         channel_choosen2 = 1 ;
         t11 = 1 ;
         avg_reward(r) = reward_vector(r) ;
```

```
      else
          channel_choosen2 = 2 ;
          t11 = 2 ;
          avg_reward(r) = reward_vector2(r) ;
      end
      reward_vector_log= 10*log10(avg_reward*20/100) ;
      cumulative_reward= cumsum(reward_vector_log)/r ;
      figure(5), plot(reward_vector_log) ;
      figure(7), plot(cumulative_reward)
   end
    channel_choosen(t) = channel_choosen1_2 ; %channel choosen
at every time slot
     flag = 0 ;
     Tm = Tm + 1 ;
     if ((channel_choosen1_2 ~=1  && channel_choosen1_2~=2)
||(channel_choosen2 ~=1  && channel_choosen2 ~=2 ))
         flag = 1 ;
     end

     end
             figure(3) , bar(prediction_forecaster2) ;

     channel_choosen_vector(r) = channel_choosen1_2 ;




      r = r+1 ;

end
clear global ;
```

caliprog.m

```
%-------------------------------------------------------------
------
%construct calibrated forecaster and output probability
distribution
%-------------------------------------------------------------
------
function [output_R_1] =  caliprog(chan_chosen,r,Tm)
% R = [1,2,3,4,5,6,7,8,9,10] ;
global vec_val_regT1  ; %initialize the vector valued regret
till time T
global vec_val_regT2 ;
global vec_val_regR1 ;
global vec_val_regR2;
global act_choosen ;
global Q ;
if(isempty(act_choosen)) %if vector is empty
    act_choosen = 1 ;
    Q = 1 ;
end

if(isempty(Q))
    Q = 1 ;
end

if(isempty( vec_val_regT1))
    vec_val_regT1 = zeros(2,21) ;
end
if (isempty(vec_val_regT2))
            vec_val_regT2  =  zeros(2,21) ;
end
% Tr = zeros(1,14) ;
% Tr = 2.^R ;
% periods = length(R) ;
% trials = 6 ;
% r = 1 ;
Tr = 2.^r ;
 C = zeros(1,21) ;


    no_joint_act_choosen_profiles = 20 ; %joint act_choosen
profile of all the players
    no_players = 2 ; %number of players
    number_of_profiles =
no_joint_act_choosen_profiles.^(no_players-1) ;
    act_choosens_of_all_players = zeros(1,21) ;
     epsilon = 2.^((-r)/(number_of_profiles+1)) ;

for j=1:21
    act_choosens_of_all_players(j) = j ; %joint act_choosen
profile
end
```

```
Ne = 20 ;

probability_player1 = zeros(2,21) ; %probability corresponding
to calibrated strategies
probability_player2 = zeros(2,21) ;
probability_player2_1= zeros(2,21) ;
probability_player2_2 = zeros(2,21) ;
for j=1:21
    probability_player1(:,j) = [(j-1)/20;1-((j-1)/20)] ;
%probability calculation (initial)
    probability_player2(:,j) = [(j-1)/20;1-((j-1)/20)] ;

end


probability_player2_1(1,:) = 1 ;
probability_player2_1(2,:) = 0 ;
probability_player2_2(1,:) = 0 ;
probability_player2_2(2,:) = 1 ;
vector_valued_r1_1 = zeros(21,21,2) ;
vector_valued_r1_2 = zeros(21,21,2) ;
vec_val_reg21 = zeros(21,21,2) ;
vec_val_reg22 = zeros(21,21,2) ;

for i=1:2
    for j = 1:21
       vector_valued_r1_1(j,j,i) = probability_player1(i,j) -
probability_player2_1(i,j) ; %calculation of vector valued
regret
       vector_valued_r1_2(j,j,i) = probability_player1(i,j) -
probability_player2_2(i,j) ;
       vec_val_reg21(j,j,i) = probability_player2(i,j)-
probability_player2_1(i,j) ;
       vec_val_reg22(j,j,i) = probability_player2(i,j)-
probability_player2_2(i,j) ;
    end
end

%% used in previous program
set11 = zeros(2,21) ;
set12 = zeros(2,21) ;
set21 = zeros(2,21) ;
set22 = zeros(2,21) ;
subset_set11 = zeros(1,21) ;
subset_set12 = zeros(1,21) ;
subset_set21 = zeros(1,21) ;
subset_set22 = zeros(1,21) ;

for j= 1:21
         set11(1,j) = vector_valued_r1_1(j,j,1) ;
         set11(2,j) = vector_valued_r1_1(j,j,2) ;
         subset_set11(j) = norm(set11(:,j)) ;
         set12(1,j) = vector_valued_r1_2(j,j,1) ;
         set12(2,j) = vector_valued_r1_2(j,j,2) ;
         subset_set12(j) = norm(set12(:,j)) ;
         set21(1,j) = vec_val_reg21(j,j,1) ;
         set21(2,j) = vec_val_reg21(j,j,2) ;
```

```
            subset_set21(j) = norm(set21(:,j)) ;
            set22(1,j) = vec_val_reg22(j,j,1) ;
            set22(2,j) = vec_val_reg22(j,j,2) ;
            subset_set22(j) = norm(set22(:,j)) ;
end
%%


output_probability_player1 = zeros(1,21) ; %output probability
distribution
output_probability_player2 = zeros(1,21) ;
output_R_1 = zeros(1,21) ;
output_R_2 = zeros(1,21) ;
act_choosen_vector = zeros(1,4) ;
pop = zeros(1,21) ;


%vec_val_regT1(:) = zeros(2,21) ;
% pop = zeros(1,21) ;






     epsilon = 2.^((-r)/(3)) ; %epsilon (for epsilon-
calibration)

    chan_chosen = chan_chosen;
        for j = 1:21
        output_probability_player1(j) = 1/20 ;
        output_probability_player2(j) = 1/20 ;
    end






    V = zeros(2,1) ;
    X = zeros(2,1) ;


        if (r==1 && Tm==1) %if first trial
%              vec_val_regT1 = zeros(2,21) ;
%              vec_val_regT2  =  zeros(2,21) ;
            pop = act_choosens_of_all_players ;
            act_choosen  =
randsample(act_choosens_of_all_players,1) ;
            Q = act_choosen ; %act_choosen corresponding to
full sample size



        elseif (r>1 && Tm==1) %if first trial in subsequent
period
            j  = 1 ;
        [val,index] = max(output_probability_player1) ;
                if(index<4)
                for k = 1:index+3
                    pop(j) = k ;
                    j = j+1 ;
                end

                elseif(index>18)
                for k = index-3 : 21
                    pop(j) = k ;
                    j = j+1 ;
                end

                else
                for k = index-3:index+3
                    pop(j) = k ;
                    j = j+1 ;
                end


                end
                  act_choosen = randsample(pop(1,1:j),1) ;
%act_choosen according to neighbourhood of probability
distribution
                  Q = act_choosen ;

            flag = 2 ;



        else

%       act_choosen  = mean(output_probability_player1) ;
%              act_choosen = round(act_choosen) ;

             U = mean(output_probability_player1) ; %calculate
average of forecaster probability distribution
             X1_1 = (output_probability_player1<=U) ;
            for i = 2:21
                if(X1_1(i) == 0)
                    act_choosen = i+1 ; %act_choosen
corresponding to index of average
                    flag1 = 1 ;
                     Q = act_choosen ;
                     break ;
                else
                    act_choosen =
randsample(act_choosens_of_all_players,1) ;
                    Q = act_choosen ;
                end
            end


%                    act_choosen  =
mean(output_probability_player1) ;
%              act_choosen = round(act_choosen) ;
%             act_choosen = 21 ;
%         Q = act_choosen ;
```

```
              if (chan_chosen == 1)
               V =
[vector_valued_r1_1(Q,Q,1),vector_valued_r1_1(Q,Q,2)] ;
            X = [vec_val_reg21(Q,Q,1),vec_val_reg21(Q,Q,2)] ;
             vec_val_regT1(:,Q) = vec_val_regT1(:,Q) + V' ;
%update regret at time T
             vec_val_regT2(:,Q) = vec_val_regT2(:,Q) + X' ;

             elseif (chan_chosen == 2)
             V =
[vector_valued_r1_2(Q,Q,1),vector_valued_r1_2(Q,Q,2)] ;
             X = [vec_val_reg22(Q,Q,1),vec_val_reg22(Q,Q,2)] ;
              vec_val_regT1(:,Q) = vec_val_regT1(:,Q) + V' ;
            vec_val_regT2(:,Q) = vec_val_regT2(:,Q) + X' ;
              end
             F = vec_val_regT1/Tr ;
             flag = 1 ;
             r1_1 = zeros(1,21) ;
             r1_2 = zeros(1,21) ;
             K = [F(1,:) F(2,:)] ;
             D = dfd(K,epsilon) ; %calculate projection

             V = zeros(1,21) ;
             X = zeros(1,21) ;
%              r1_1 = zeros(1,21) ;
%              r1_2 = zeros(1,21) ;
%              Y1 =
dot(vec_val_regT1(1,:),subset_set11(:))./sum(subset_set11(:).*
subset_set11(:)) ;
%              C1 = bsxfun(@times,Y1,subset_set11(:)) ;
%              Y2 =
dot(vec_val_regT1(2,:),subset_set12(:))./sum(subset_set11(:).*
subset_set11(:)) ;
%              C2 = bsxfun(@times,Y2,subset_set12(:)) ;
%              C = [Y1 Y2]' ;
%              V = zeros(2,21) ;
%              X = zeros(2,21) ;
              for i = 1:21
                  V(1,i) = vector_valued_r1_1(i,i,1) ;
                  V(2,i) = vector_valued_r1_1(i,i,2) ;
                  X(1,i) = vector_valued_r1_2(i,i,1) ;
                  X(2,i) = vector_valued_r1_2(i,i,2) ;
              end
             T = K - D ;
             temp1 = [V(1,:) V(2,:)] ;
             temp2 = [X(1,:) X(2,:)] ;
             loss_incurred = zeros(1,21) ;
             V = zeros(21,2) ;
%             regret_vecvalued1 =
probability_player1.*(temp1+temp2);
             for i = 1:21
             loss_incurred(i) =
probability_player1(1,i).*(dot(T,temp1)) +
probability_player1(2,i).*(dot(T,temp2)) ; %calculate loss
             end
             %                r1_1(1,:)=
dot(T(1,:),temp1(1,:)) ;
```

```
%                   r1_1(2,:) = dot(T(2,:),temp1(2,:));
%                   r1_2(1,:) = dot(T(1,:),temp2(1,:)) ;
%                   r1_2(2,:) = dot(T(2,:),temp2(2,:));

%               if (chan_chosen == 1)
%
%                   r1_1(1,Q)= dot(T(1,:),temp1(1,:)) ;
%                   r1_1(2,Q) = dot(T(2,:),temp1(2,:));
%               else
%                   r1_2(1,Q) = dot(T(1,:),temp2(1,:)) ;
%                   r1_2(2,Q) = dot(T(2,:),temp2(2,:));
%               end

%               regret2_1= zeros(1,21) ;
%               regret2_2 = zeros(1,21) ;
%               Y1 =
dot(vec_val_regT2(1,:),subset_set21(:))./sum(subset_set21(:).*
subset_set22(:)) ;
%               C1 = bsxfun(@times,Y1,subset_set21(:)) ;
%               Y2 =
dot(vec_val_regT2(2,:),subset_set22(:))./sum(subset_set21(:).*
subset_set22(:)) ;
%               C2 = bsxfun(@times,Y2,subset_set22(:)) ;
%               C = [Y1 Y2]' ;
%               V = zeros(2,21) ;
%               X = zeros(2,21) ;
%               V(1,:) = vec_val_reg21(Q,:,1) ;
%               V(2,:) = vec_val_reg21(Q,:,2) ;
%               X(1,:) = vec_val_reg22(Q,:,1) ;
%               X(2,:) = vec_val_reg22(Q,:,2) ;
%               T = vec_val_regT2 - C ;
%               temp1 = [V(1,:); X(1,:)] ;
%               temp2 = [V(2,:); X(2,:)] ;
%               if (chan_chosen == 1)
%
%                   regret2_1(1,Q)= dot(T(1,:),temp1(1,:)) ;
%                   regret2_1(2,Q) = dot(T(2,:),temp1(2,:));
%
%               else
%                   regret2_2(1,Q) = dot(T(1,:),temp2(1,:)) ;
%                   regret2_2(2,Q) = dot(T(2,:),temp2(2,:));
%               end
%
%
%               loss2 = regret2_1.*probability_player2(1,Q) +
regret2_2.*probability_player2(2,Q) ;
            rounds = 10;
            loss_factor = 1./(1+sqrt(2*log(Ne))/rounds);
            s1 = 0 ;
            s2 = 0 ;
            loss_factor.^loss_incurred ;
            for j = 1:rounds %multiplicative weight algorithm

%                   if (chan_chosen == 1)
```

```
                s1 =
sum(output_probability_player1.*(loss_factor.^loss_incurred))
;
                output_probability_player1 =
output_probability_player1.*(loss_factor.^loss_incurred)/s1 ;
%                s2 =
sum(output_probability_player2.*(loss_factor.^loss2(:))) ;
%                 output_probability_player2 =
output_probability_player2.*(loss_factor.^loss2(:))/s2 ;
                flag1 = 1 ;

%                 else
%
%                 output_probability_player1(Q) =
output_probability_player1(Q)*(loss_factor.^loss_incurred(2,Q)
)/s1 ;
%                 s1 = s1 +
output_probability_player1(Q).*(loss_factor.^loss_incurred(2,Q
)) ;
%                 s2 = s2 +
output_probability_player2(Q)*(loss_factor.^loss2(2,Q)) ;
%                 output_probability_player2(Q) =
output_probability_player2(Q)*(loss_factor.^loss2(2,Q))/s2 ;




            end
            flag = 3 ;

        vec_val_regR1 = vec_val_regT1 ;
        vec_val_regR2 = vec_val_regT2 ;
        output_R_1 = output_probability_player1;
%         output_R_2 = output_probability_player2 ;



       % figure(1), bar(Z) ; %ylim([0.000 0.020]) ;

       % figure(2), bar(output_R_1) ;
        if(abs(vec_val_regR1) > epsilon) %condition of
calibration
            r = 1 ;
       end
%        act_choosen  = mean(output_probability_player1) ;
%              act_choosen = round(act_choosen) ;
%             act_choosen = 21 ;
%         Q = act_choosen ;
%              act_choosen  = mean(output_probability_player1)
;
%              act_choosen = round(act_choosen) ;
             U = mean(output_probability_player1) ;
             X1_1 = (output_probability_player1<=U) ;
            for i = 2:21
                if(X1_1(i) == 0)
                    act_choosen = i+1 ;
                     Q = act_choosen ;
                    break;
               else
                   act_choosen =
randsample(act_choosens_of_all_players,1) ; %refer previous
section
                   Q = act_choosen ;
               end
           end

%              cdf = cumsum(output_probability_player1) ;
%              U = rand ;
%              X = sum(cdf<=U) ;
%             act_choosen = X ;
%         Q = act_choosen ;

%  act_choosen = random(foreceaster_probability_player1) ;
%         Q = act_choosen ;

        end
        Q = act_choosen ;


end %end
```

reward_calculation.m

```
%----------------------------------
%reward calculation code snippet
%---------------------------------
sum1 = zeros(M,M);
sum2 = zeros(M,M) ;
reward = zeros(1,3) ;
P = [100 500 800] ;
sum_final = zeros(M,M) ;
sum_final_2 = zeros(M,M) ;
X = zeros(size(M,M)) ;
Y = zeros(size(M,M)) ;
P1 = 0 ;
k = 3 ;
reward_3 = zeros(size(Hi_misor1)) ; %reward vector
SINR_1 = zeros(1,1) ;
SINR_3 = zeros(1,1) ;
for i = 1:2
            X = Hi_misor1(:,:,2,1) ;
        reward(i)=
P(i)*(real(trace(Hi_misor1(:,:,i,j)))))./((P(channel_choosen2)*
(real(trace(X))))+((noise_variance1))); %calculate reward as
per SINR formula
end
SINR_1 = [reward(1), reward(2)]; %return reward to MAB program
```

projection.m

```
%----------------------------------------------
%calculate projection of one vector on another
%----------------------------------------------
function [optimal_solution] = dfd(vec_valued_regret,eta)
mu = zeros(1,1) ;
temp = zeros(2,41) ;
M = zeros(1,1) ;
    for j = 1:42
    if (vec_valued_regret(j) < 0)
        vec_valued_regret(j) = -vec_valued_regret(j) ; %check
sign of vector store it and make it positive
        sign(j) = -1 ;
    else
        sign(j) = 1 ;
    end
    flag = 0 ;
end
N = 42 ; %size when vector is stacked
optimal_solution = zeros(1,42) ;
X = vec_valued_regret ; %temporary storage variable for
original value
Nbar = N ;
%for j = 1:2
%for i = 1:40


if (sum(vec_valued_regret) <= eta) %if all values lesser than
epsilon
        optimal_solution= sign.*vec_valued_regret;%use
original vector
        flag = 1 ;
    else
        flag = 3 ;
         Nbar = N ; %else
         vec_valued_regret = sort(vec_valued_regret,'descend')
; %in descending order
             vec_valued_regret(:,N+1)= -inf ;
while(1)
    if(vec_valued_regret(:,Nbar) >
vec_valued_regret(:,Nbar+1)) && (vec_valued_regret(:,Nbar) >
(sum(vec_valued_regret(1,1:Nbar)) - eta)/Nbar) %and check for
exit conditon
        flag = 2 ;

%            if (vec_valued_regret(j,i) <
vec_valued_regret(j,i+1))
%             temp = vec_valued_regret(j,i) ;
%             vec_valued_regret(j,i) =
vec_valued_regret(j,i+1) ;
%             vec_valued_regret(j,i+1) = temp ;
%             end

                        mu = (sum(vec_valued_regret(1,1:N)) -
eta)/Nbar ; %loop until condition is satisfied
                        for i = 1:42 %update mu and break from
loop
                        K(i) = minus(vec_valued_regret(i),mu)
;
                        end
                        if (size(K) >= [1 0])
                            flag = 6 ;
                         % optimal_solution(i) = K(1,1) ;
                        end
                        optimal_solution = sign.*K ; %before
exiting update optimal solution
                        flag = 4 ;
                         vec_valued_regret = X ;
                        break;
                     else  %if exit condition not satisfied
                         Nbar = Nbar-1 ; %update Nbar and next
loop

                         flag1 = 5 ;
                     end

end
end

end    %end
```