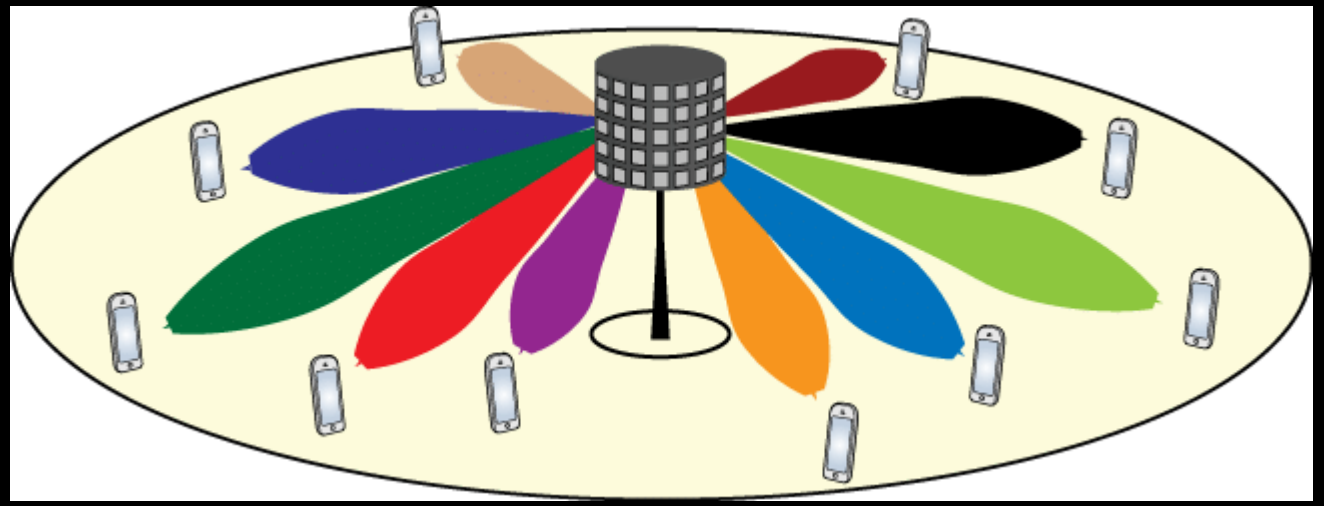
Joint Precoding and Power Control Design in Massive MIMO: A Game Theoretic Approach

Anurag Pallaprolu, Mohammed Abdelghany

ECE Department
University of California, Santa Barbara

Motivation

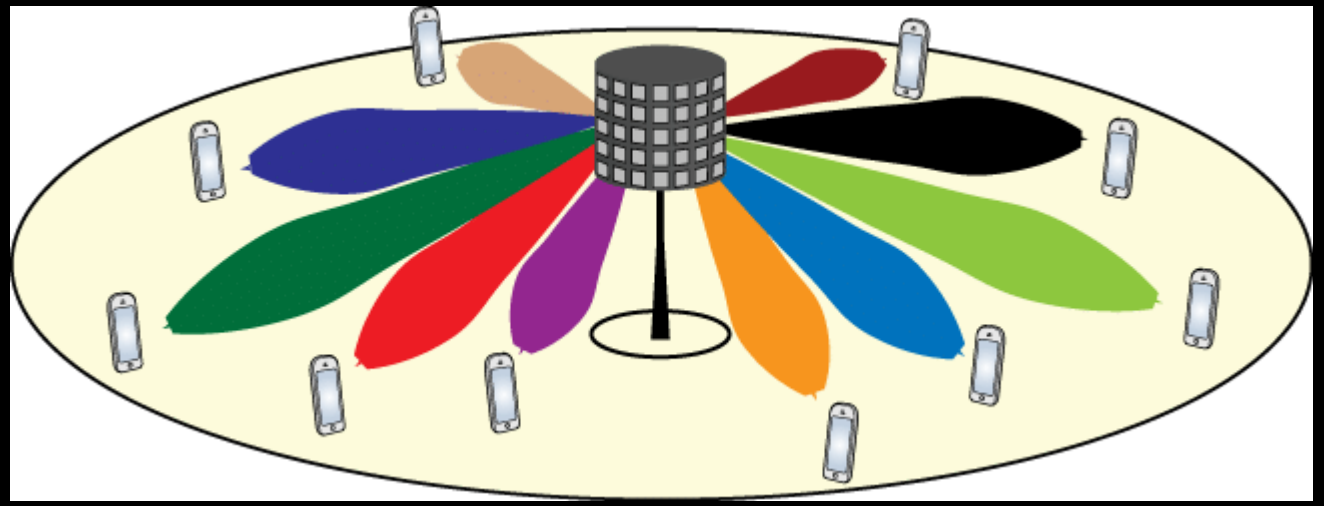


- To counter interference driven SINR reduction, subscriber mobile stations are liable to use maximum TX power as a worst case strategy [see 1, 2] that leads to some of them receiving lower overall link quality.

$$SINR_k^u = \frac{a_{kk}p_k}{\sigma^2 + \sum_{\substack{i=1 \\ i \neq k}}^{K} a_{ik}p_i}$$

$$SINR_k^d = \frac{b_{kk}q_k}{\sigma_k^2 + \sum_{\substack{i=1 \\ i \neq k}}^{K} b_{ki}q_i}$$

Motivation



- To counter interference driven SINR reduction, subscriber mobile stations are liable to use maximum TX power as a worst case strategy [see 1, 2] that leads to some of them receiving lower overall link quality.

- However, if the **base station can provide feedback to the users through a capacity-limited noiseless channel**, we can design power usage policies that helps us attain Pareto-efficiency/Socio-optimal Nash equilibrium.

Outline

- Game setup: Definitions of the players, their action spaces and the corresponding utility functions. In this case we always have two utility functions and therefore is a multi-objective optimization game.
- Discussions: We deal with four cases which encompass interesting variations of players and channel behavior.
- Results: As we explore the scenarios, we show the variations of the Price of Anarchy (PoA)
- Future prospects: Sneak peek into where one could go from here.

Game Setup

- Players: K mobiles and a base station:
  - Mobiles Set of Actions: $p_k \in [0, P_o], \ \forall k \in [1, K]$
  - Base station Set of Actions: $q_k \in [0, P_{\text{tot}}], \ s.t. \sum_i q_i = P_{\text{tot}}, \ \forall k \in [1, K]$
- Player k utility function: $\log_2(1 + SINR_k^u) + \log_2(1 + SINR_k^d)$
- Base station utility function: $\max_{q_k \forall k} \min \left(\min_{k \forall k} SINR_k^u, \min_{k \forall k} SINR_k^d \right)$

Price of Anarchy

PoA(X) = (Worst uplink SINR when Pareto-efficient)/(Worst uplink SINR for a scenario X)

$$SINR_k^u = \frac{a_{kk}p_k}{\sigma^2 + \sum_{\substack{i=1 \\ i \neq k}}^{K} a_{ik}p_i}$$

Beamforming and Power Allocation

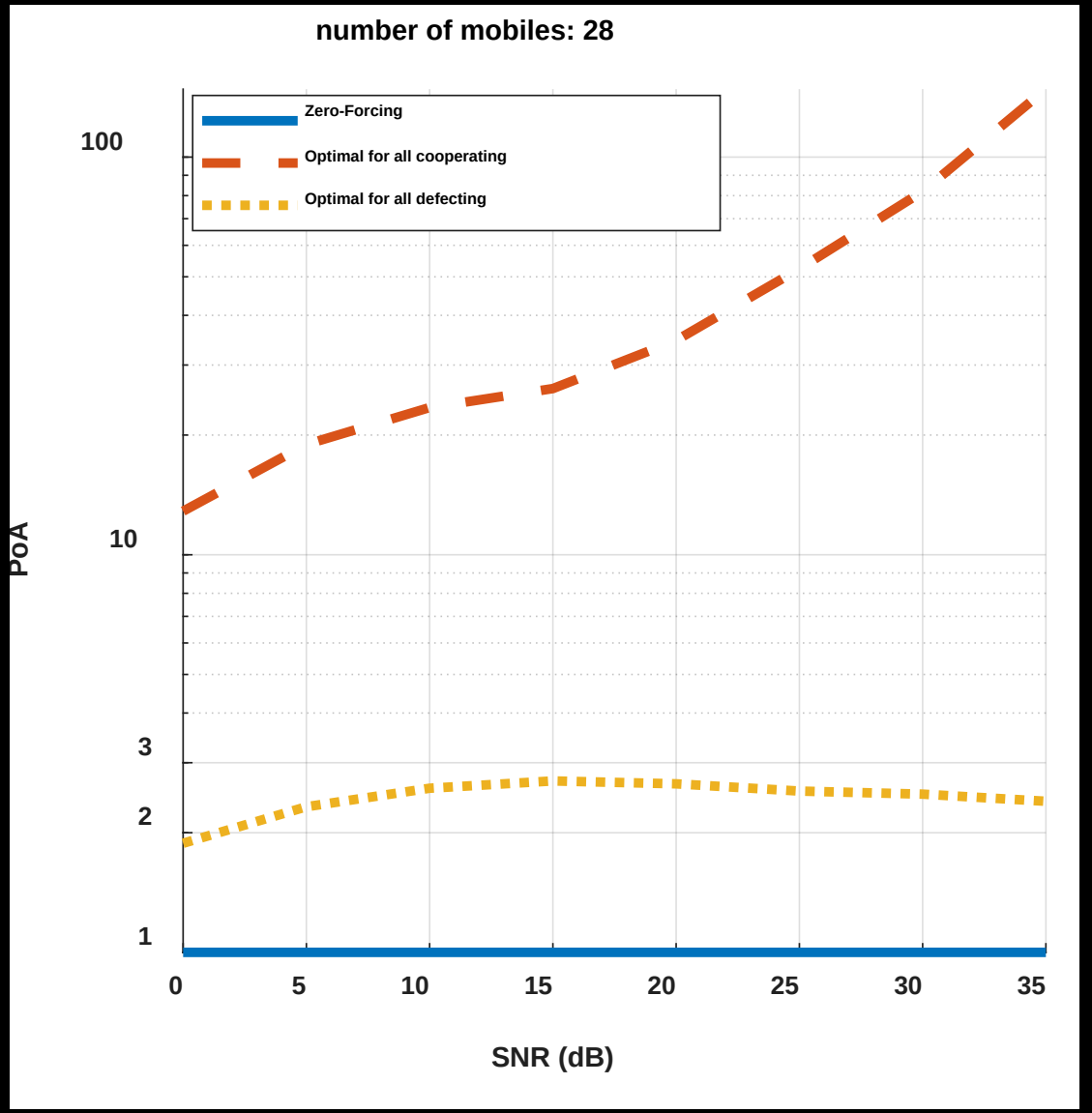
- Beamforming is like constructing the road.
- Power allocation is equivalent to controlling the traffic on that already constructed world.
- We pick a beamforming method beforehand, then setup a game to establish social equilibrium.

No-Cooperation Scenario

- The Price of Anarchy is minimum (= 1) if zero-forcing beamforming is deployed.
- This is because that zero-forcing beamforming suppress the interference between the mobiles
- It designs the matrix A to be diagonal:

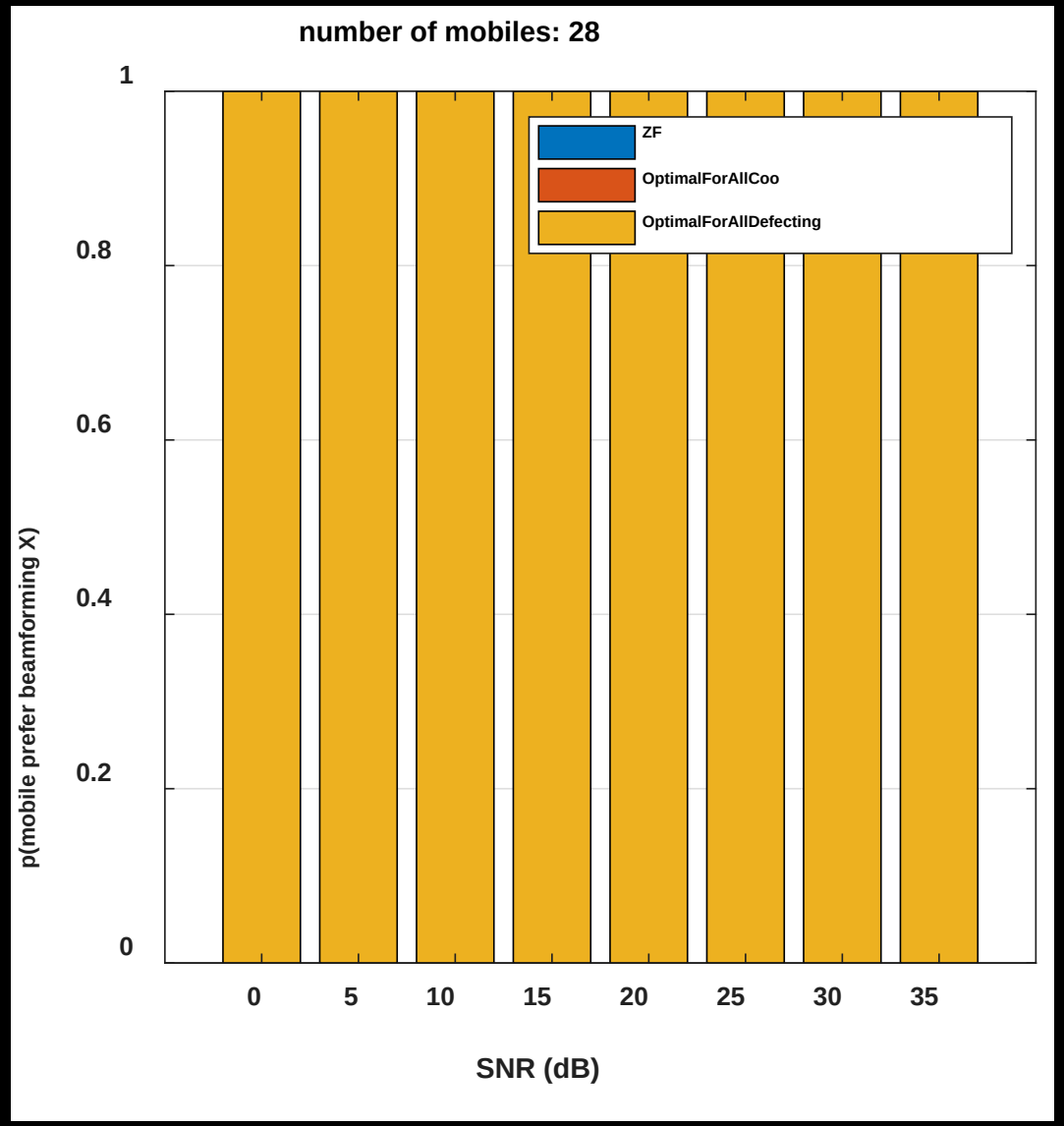
$$SINR_k^u = \frac{a_{kk}p_k}{\sigma^2 + \sum_{\substack{i=1 \\ i \neq k}}^{K} a_{ik}p_i}$$

- As a result, non-cooperative manner (everyone send with his/her maximum power) becomes the social optimum.



No-Cooperation Scenario

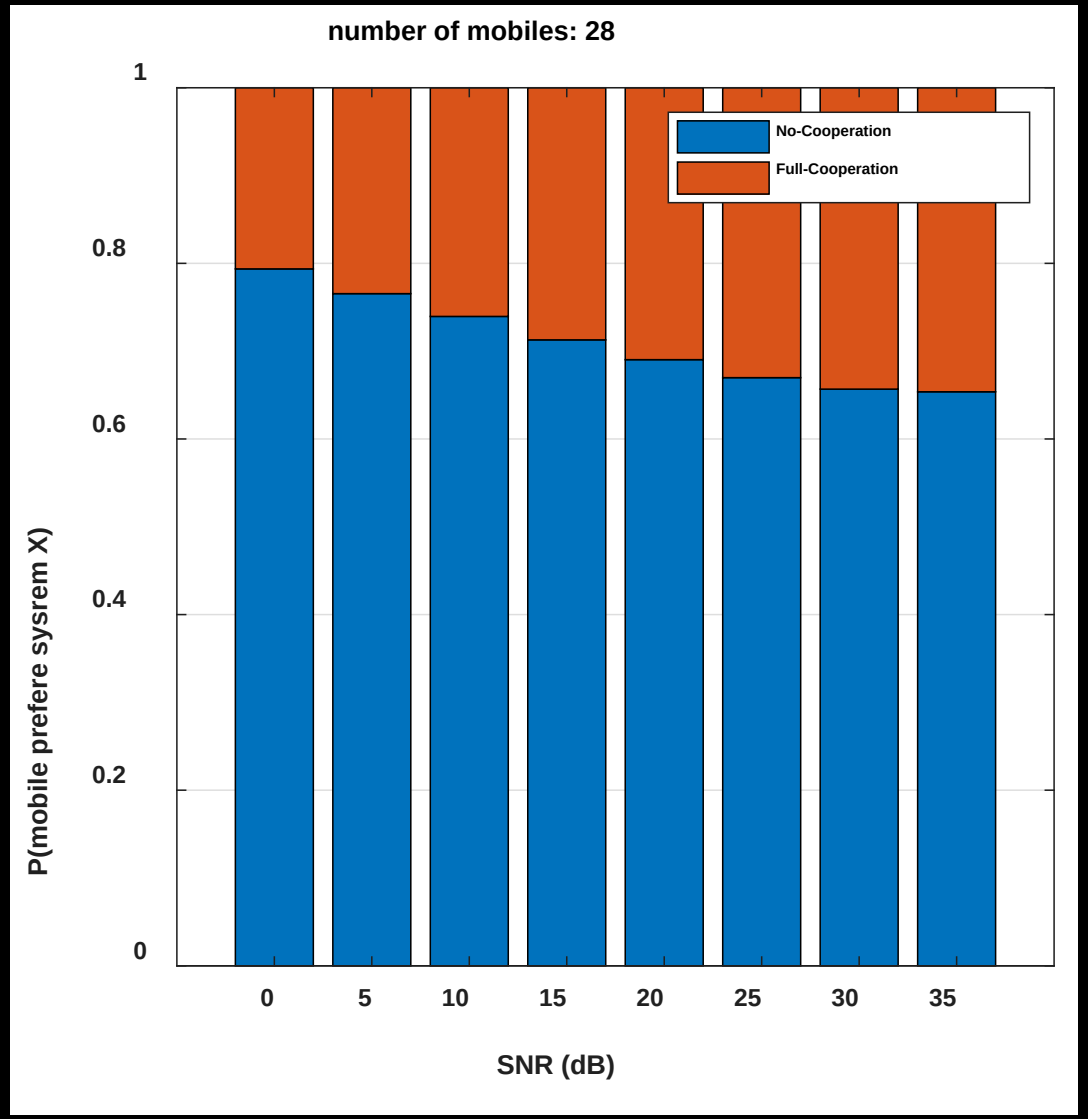
- Nevertheless, Not a single mobile station like it.
- All mobiles prefers to live in a bit chaotic world, i.e., PoA>1, that takes care of them.



Full-Cooperation Scenario

- Not all the mobile stations prefer to live in the utopian world, where the system takes care of the weakest mobile.
- This is because their utility function:

$$\log_2(1 + SINR_k^u) + \log_2(1 + SINR_k^d)$$



Mixed Policies

The rigidity of the pure policy (always defect or always abide) makes it unrealistic as a model of how actual subscribers would probably behave. If we instead assume that they defect uniformly, we see the following variation in the approach to equilibrium

```
case 'C2'
   for quantum=linspace(0.1, 0.4, 25)
    %% Start with defecting because no incentive
    PdUpperLimit=ones(nUsersTmp,1); % Prealocation
    PuLowerLimit=zeros(nUsersTmp,1); % Prealocation
    SINRdPromise=zeros(nUsersTmp,1); % Prealocation
    SINRuPromise=zeros(nUsersTmp,1); % Prealocation
    % Optimal Downlink Game at Base station if every user cooperate
    [~,SINRdOptimal] = DLPowerAlloc_coreFn('C',B,Init.sigma(i_sigma).^2,Init.Nx,Init.Nx*ones(nUsersTmp,1),zeros(nUsersTmp,1),zeros(nUsersTmp,1));
    % Optimal Uplink Game at Base station if every user cooperate
    [~,SINRuOptimal] = ULPowerAlloc_coreFn('C',A,Init.sigma(i_sigma).^2,ones(nUsersTmp,1),zeros(nUsersTmp,1),zeros(nUsersTmp,1));
    defectingUsers=true(nUsersTmp,1);
    PuUpperLimit=ones(nUsersTmp,1);
    NEflag= false;

    probOfDefecting=quantum*ones(nUsersTmp,1);rand(nUsersTmp,1);
    % Feedback starts
    PoA=[];
    iter = 0; %TX-RX Cycle Count
    stabilizer = 0; %Stable equilibrium check
    while(iter < 100)
```

Mixed Policies

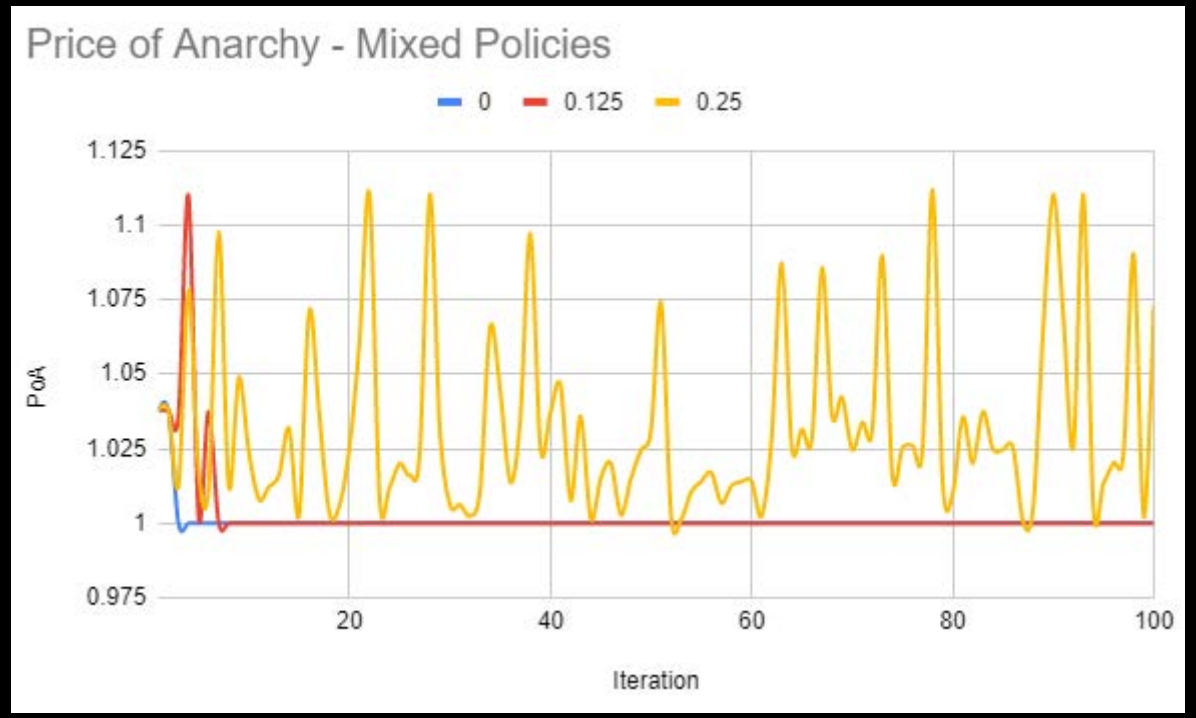
The rigidity of the pure policy (always defect or always abide) makes it unrealistic as a model of how actual subscribers would probably behave. If we instead assume that they defect uniformly, we see the following variation in the approach to equilibrium

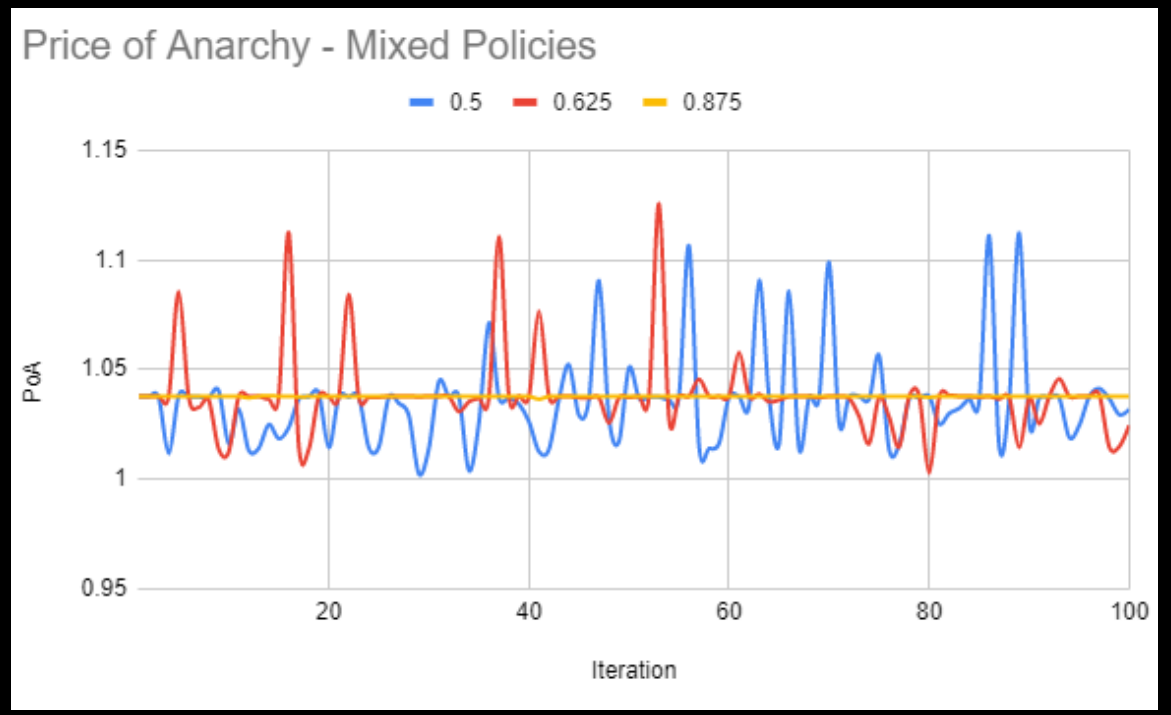
```
%% Uplink Game Optimization start
Pu=PuUpperLimit;
%% Actual Uplink SINR
SINRu=(diag(A).*Pu)./(Init.sigma(i_sigma).^2+A'*Pu-diag(A).*Pu);
%% Base station applies it policy
PdUpperLimit(defectingUsers)=0;
PdUpperLimit(~defectingUsers)=Init.Nx;
PuLowerLimit(defectingUsers)=1;
PuLowerLimit(~defectingUsers)=0;
%% Downlink Game start
% for cooperating users
[Pd,SINRd] = DLPowerAlloc_coreFn('C',B,Init.sigma(i_sigma).^2,Init.Nx,PdUpperLimit,zeros(nUsersTmp,1),defectingUsers);
SINRdPromise(~defectingUsers)=SINRdOptimal(~defectingUsers);
SINRdPromise(defectingUsers)=0;
%% Base station feedback
%% Virtual uplink
[PuAdvised,SINRtmp] = ULPowerAlloc_coreFn('C',A,Init.sigma(i_sigma).^2,ones(nUsersTmp,1),PuLowerLimit,defectingUsers);
SINRuPromise(~defectingUsers)=SINRtmp(~defectingUsers);
SINRuPromise(defectingUsers)=SINRuOptimal(defectingUsers);
%% users decide to follow or not
defectingUserstmp=Init.uUtility(SINRuPromise,SINRdPromise)<Init.uUtility(SINRu,SINRd);
randToss=rand(nUsersTmp,1);
defectingUserstmp(defectingUserstmp)=randToss(defectingUserstmp)<probOfDefecting(defectingUserstmp);
PuUpperLimit(~defectingUserstmp)=PuAdvised(~defectingUserstmp);
PuUpperLimit(defectingUserstmp)=1;
% Condition to get outside (does NE has been reached?)
```

```
iter = iter + 1;
disp({'C2', iter, quantum,
    min(SINRuOptimal)./(min(SINRu))})
if (min(SINRuOptimal)./(min(SINRu)) == 1)
    stabilizer = stabilizer + 1;
end
if (stabilizer == 5)
    break
end
```

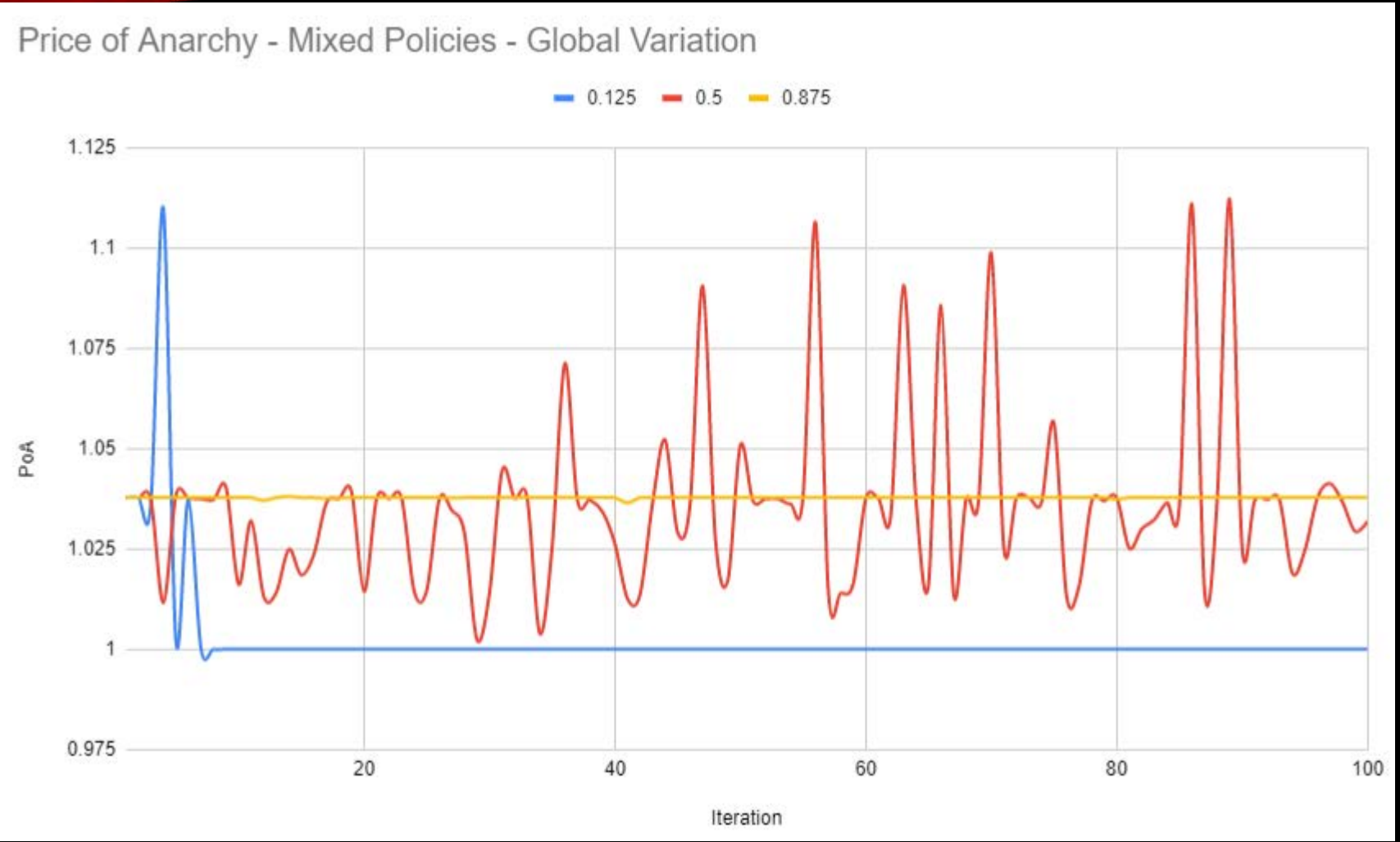
Case C2 - Mixed Policies

The rigidity of the pure policy (always defect or always abide) makes it unrealistic as a model of how actual subscribers would probably behave. If we instead assume that they defect uniformly, we see the following variation in the approach to equilibrium





Global Variation



We see a **bistable behavior**, that is, if the probability of defecting is less than a threshold we will converge to the social equilibrium in very few iterations or else we witness an oscillatory behavior that instead converges to Case B

Search For Transition

The experiments involved a bisection based search where we zero in on an interval to the left of which we see stable behavior but to the right we witness oscillations.

It is not quite clear if we can say that 28% is the threshold since at 30% we once again see the transient behavior.

Search For Transition

However, for probabilities greater than 30% it is hard to see such a transient convergence and this gets even harder as we increase the percentage beyond 40%.

Obviously, at near 100% we will return to Case B as explained earlier.

Fault Tolerance Through Consensus?

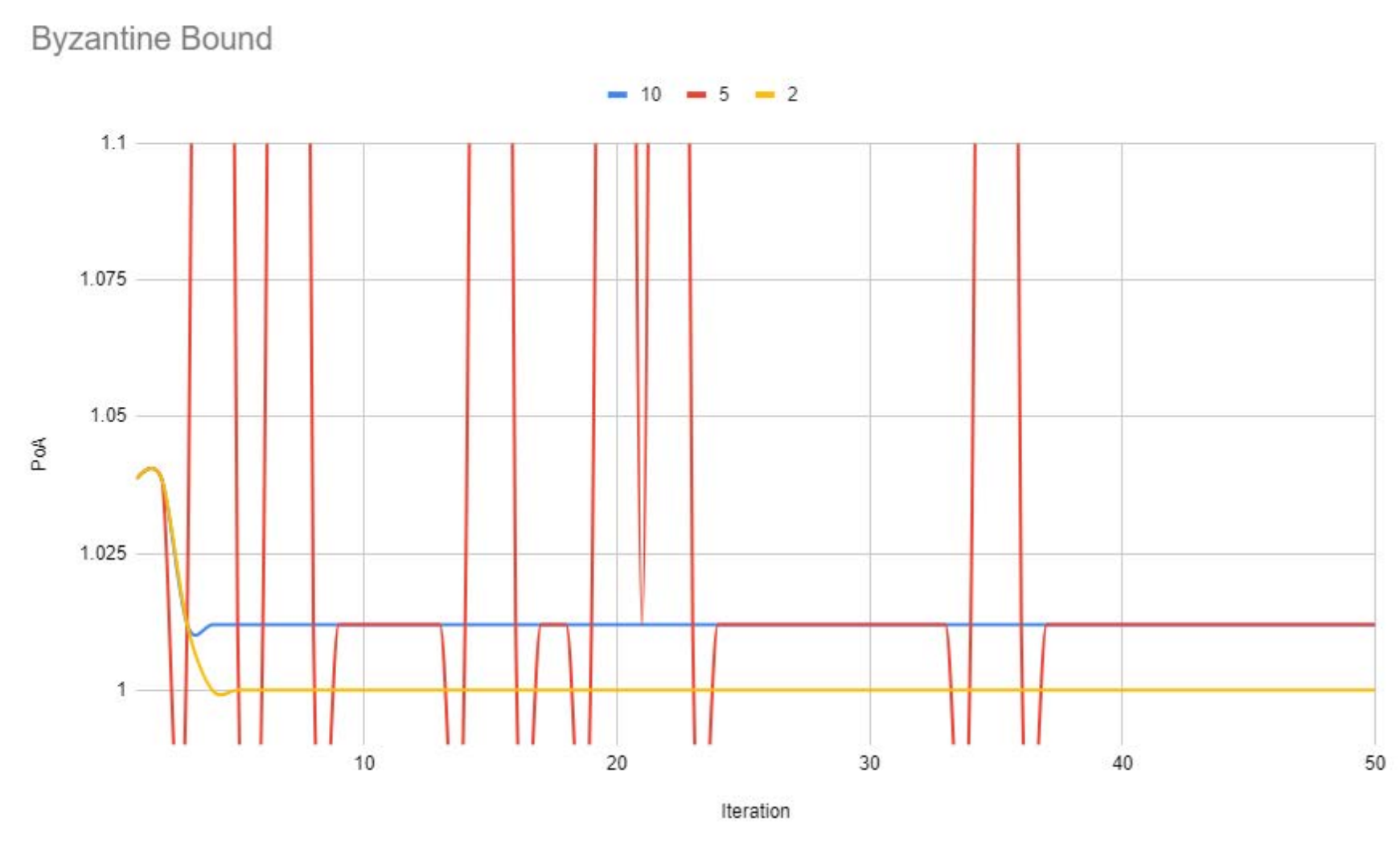
- Is there a way to achieve social equilibrium for higher rates of defecting?
- If we allow the subscribers to cooperate, we can indeed ensure convergence to the social optimum by implementing a consensus based decision making scheme popularly known as **Byzantine Fault Tolerance**
- We allow users to share their decisions with each other so that each of them can take the majority decision as the collective one. Clearly, since they would either all choose to defect or abide, this should lead to lesser uncertainty
- But what if a certain user does not follow a decision he communicates to his peers? We call such users **“traitors”.**
- This algorithm is popular in distributed systems research and has many applications: blockchain consistency, NFS protocol fault tolerance et cetera.

Fault Tolerance Through Consensus

```
% Construct the consensus matrix. Initially it will have all identical copies in an utopian world where there are n
% traitors.

consensus = [];
for user=1:length(defectingUserstmp)
    consensus = [defectingUserstmp';consensus];
end

% We then define the traitors.

traitors = [1:traitor_count];
soc = size(consensus);

% Each traitor will randomly decide to send the wrong message to the other users thus modifying the state of the
% consensus matrix.

for t=1:length(traitors)
    for j=1:soc(1)
        %x = rand(1, soc(1));
        %if(x(j) > 0.5)
            consensus(j, traitors(t)) = rem(j, 2);
        %else
        %    consensus(j, traitors(t)) = false;
        %end
    end
end

% Now each user will look at his/her own rows and take the majority consensus as their final decision.
% NOTE: This should not be confused with their independent probabilities of defecting which is already considered
% before the consensus voting starts.

defectingUserstmp = logical(mode(consensus, 2));

% End of Byzantine traitoring.
```

Implementation is the same as case C2 except that before we find out which users actually defect or abide we include a consensus voting round keeping in mind the possibility of traitors.

Lamport-Byzantine Bound



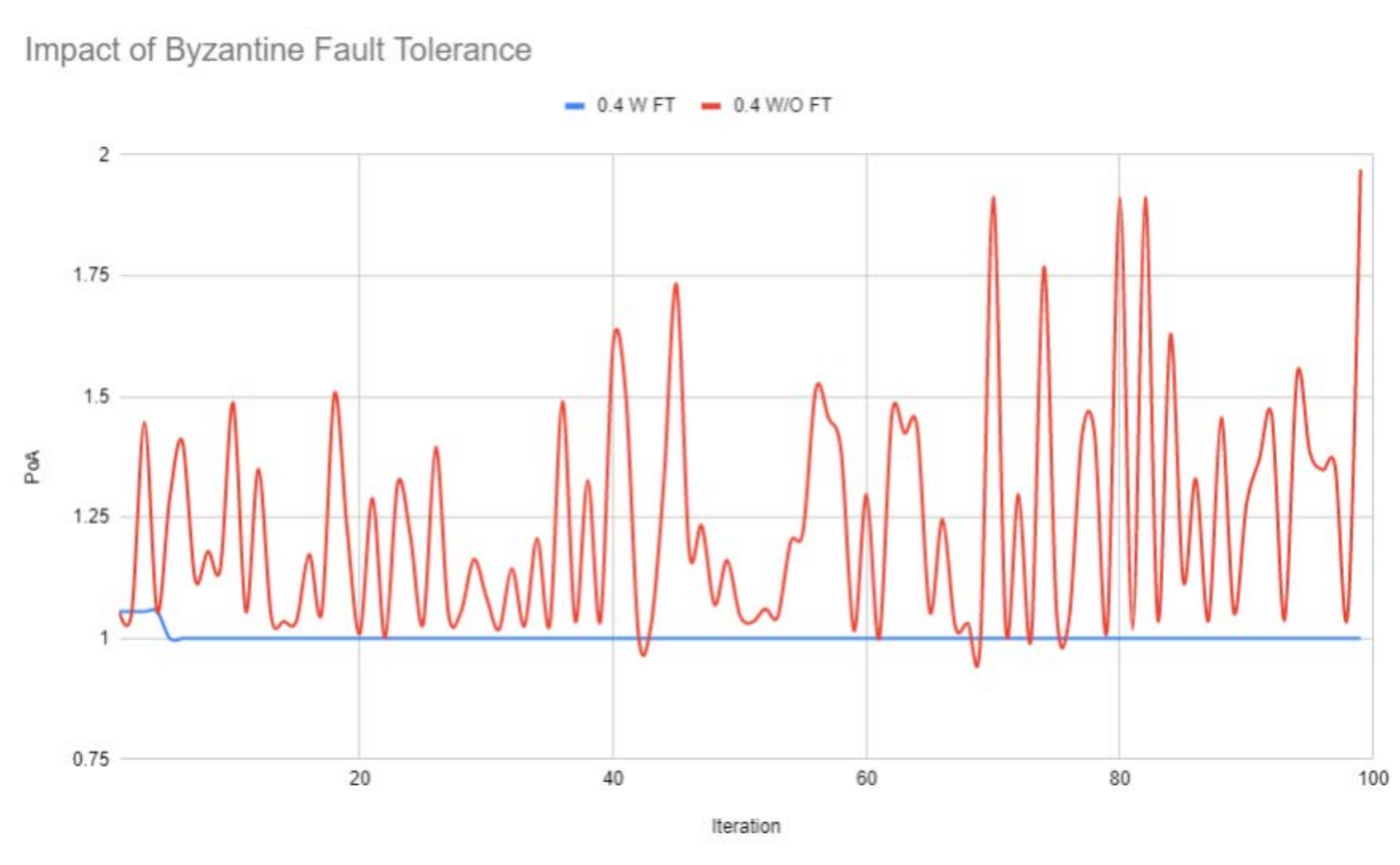
The plot represents the movement of social equilibrium when the BFT algorithm is implemented for a 50% defect rate.

The different plots describe different number of traitors.

Leslie Lamport[5] famously proved that **if more than 1/3rd of the players are traitors we cannot achieve perfect consensus.**

We can observe the same in this example as well (N=16).

Comparison



However, when the number of traitors are below the Lamport bound, we see a staggering improvement in convergence.

It is to be kept in mind that this is **not** a fully cooperative setting since the base station does **not receive** any information about the consensus directly from the subscribers.

It is however able to indirectly influence the convergence through feedback.

Further Improvements

- CVX Runtime Optimization: We saw many issues with convergence initially due to the Nesterov interior point iterations[3] but MA has made progress with a new fixed-point seeking convergence routine that runs recursively instead[4]. **(Already done)**

- Castro-Liskov BFT: M. Castro and B. Liskov [6] came up with another strategy which incorporates cryptographic verification instead of blind consensus in BFT that was implemented on a NFS protocol.

- Fully Cooperative Game: We could consider a case D which involves sending the consensus as a feedforward message to the base station. The base station can have a smoothing function that depends on number of active connections (e^(-N), 1/N, …) and will then have to maximize this as well for efficient business.

References

- 1: G. Scutari, D. P. Palomar, and S. Barbarossa, “Optimal Linear Precoding Strategies for Wideband Noncooperative Systems Based on Game Theory—Part I: Nash Equilibria,” *IEEE Transactions on Signal Processing*, vol. 56, no. 3, pp. 1230–1249, 2008.
- 2: D. Palomar, J. Cioffi, and M. Lagunas, “Joint TX-RX beamforming design for multicarrier mimo channels: a unified framework for convex optimization,” *IEEE Transactions on Signal Processing*, vol. 51, no. 9, pp. 2381–2401, 2003.
- 3: Y. Nesterov and A. Nemirovsky, “Interior-point polynomial methods for convex programming”, SIAM Studies Appl. Math., Vol 13, 1994.
- 4: Ami Wiesel, Yonina C. Eldar, and Shlomo Shamai, “Linear Precoding via Conic Optimization for Fixed MIMO Receivers.”
- 5: L. Lamport, R. Shostak, M. Pease, “The Byzantine Generals Problem”, *ACM Transactions on Programming Languages and Systems,* Vol. 4, No. 3, July 1982, pp. 382-401.
- 6: M. Castro, B. Liskov, “Practical Byzantine Fault Tolerance”, *Proc. of Third Symposium on Operating Systems Design and Implementation,* New Orleans, USA, February 1999.

Thank You!