CS-765 Assignment 3 Report

200050019 - Bagathi Shiv Kiran
200050020 - Bale Teja Rama Chandra Murthy
200050075 - Moganti Harshadeep

Part-A:

1. **Sybil Attack:** In our equation to calculate the weights for the voters, weight of the voter is directly proportional to the stake of the voter. Thus it is more beneficial even for the attacker to have all his stake stacked up at one place, rather than distributing it. Thus, it prevents the attacker from performing a Sybil attack.
2. **Method to evaluate or re-evaluate the trustworthiness of voters:** We evaluate the voters based on the correctness of their votes in the voting. We are employing the following method to evaluate and re-evaluate the trustworthiness of voters after each time a voter casts his vote:

```
for vote in news_item.votes:
    if vote.vote == true_value:
        vote.voter.alpha += 1
    else:
        vote.voter.beta += 1
    vote.voter.reputation = beta.mean(vote.voter.alpha, vote.voter.beta)
    voter_reputation[index][vote.voter.id] = vote.voter.reputation
```

Here alpha(α) signifies the total number of correct votes that are cast, beta(β) signifies the number of votes cast which are incorrect. Reputation is updated as follows:

$$Reputation = \alpha \div (\alpha + \beta)$$

Through our Simulation, we have shown that our algorithm leads the Reputation scores to converge as expected.

Trustworthiness is stored in a dictionary with keys corresponding to different fields.

3. **More trustworthy voters should be given more weight:** Weights for the votes of the voters is directly proportional to the trustworthiness factor, calculated as stated above, and stake of the particular voter.(To prevent Sybil attack).We have calculated the weights for our voters as follows:

```
weighted_sum = sum(vote.vote * vote.voter.reputation * vote.voter.stake for vote in self.votes)

# Modify the total reputation to include the stake
total_reputation = sum(vote.voter.reputation * vote.voter.stake for vote in self.votes)

final_result = weighted_sum / total_reputation
```

Weight of each vote is :

$$weight = \sqrt{stake\ of\ the\ voter} \times reputation\ of\ the\ voter$$

Here, we put a square root to control the rich.(We can use any function f, f" <0) Everytime a voter votes correctly, it is not just his reputation that increases, but also his stake. Thus if the voter is more trustworthy, then the weight for his vote increases. Weights are stored in a dictionary with keys corresponding to different fields.

4. **Rational voters are to be incentivised:** The client who calls for the function to verify the news article has to pay some fee and even the voters who want to vote have to lock their stake. We impose a condition that a fraction(f) of the locked stake of all the voters who vote incorrectly will be added to the verification fee provided by the client and distributed equally between all the voters who voted correctly. (Serves as incentive for rational voters)

```
else{
    voter.incorrect += 1;
    b = voter.incorrect[article.area];
    fee += article.votes[i].stake/10;
    voter.stake = (vote.stake*9)/10;
```

```
if(article.votes[i].vote == truth){
    voter.stake += fee/correct_voters;
```

We also propose that the highly trustworthy voters(criteria will be having direct proportionality with the rating as well as number of votes cast) will form some sort of committee and send some articles(whose truth value is known to them) for all the remaining voters as a test, and the punishment will be harsher for those who voted incorrectly(f' fraction of stake will be taken; f'>f). The reasons for choosing to have this harsher punishments will be explained in Part C. Voters not in the committee should not know if they get harsher punishments or not.

5. **Uploading a news item:** We are using the hash of the content to create an identifier for the news article, the client should also provide the field of the article, incentive for the voters to verify the article.
6. **Bootstrapping:** Initially when a user wants to register in this DApp, they will be asked to choose a field which they think they are most trustworthy. We initially assign the reputation score of all the voters to 1 and update the reputation scores as:

$$Reputation\ score\ =\ \alpha\ \div\ (\alpha\ +\ \beta)$$

where α signifies the number of votes which are correctly cast by this voter and $\alpha\ +\ \beta$ signifies the total number of votes that are cast by this voter. By using this method, we see that our Reputation score converges as expected.(From our results in the simulation done in Part-C, which are explained below.)

Part C :

By using our algorithm ,stated above, when we are simulating for q = 0.5, we found that malicious users are winning by having the highest reputation scores. This is expected because the malicious users comprise 50 percent of our network here and thus their vote becomes the truth for the article with highest probability.

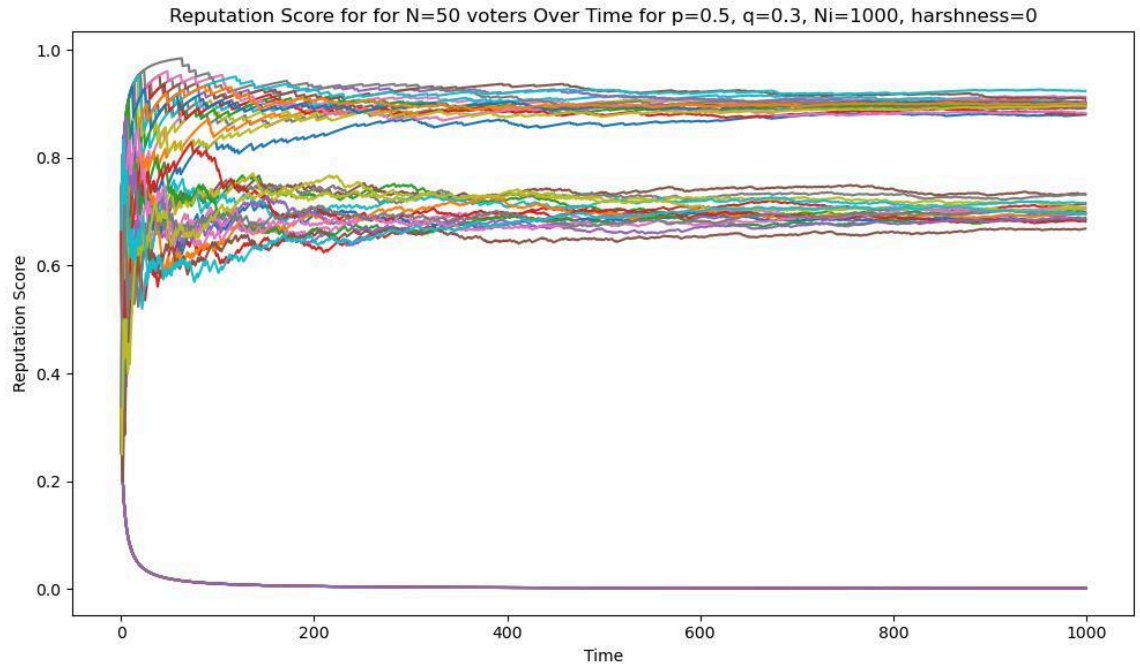
So we added harshness as an added feature to facilitate punishment of the malicious. When we have harshness (set appropriately) then even up to q = 0.8 (or more) our trustworthy DApp voters score converges to higher values, making our system more resilient. What this harshness does is, it takes harshness as an input parameter and it increases/decreases your correct/incorrect scores based on your answer by the harshness factor instead of changing them by 1.

Thus it gives a boost to the honest reputation score whenever their answer wins and decreases the malicious reputation score in a similar way. Ideally, we would want more trustworthy users to form a committee of some sort and randomly insert an article of which they know the truth and harshly evaluate the rest of the voters. This should be done in such a way that the voters not in the committee should never know if the evaluation is harsh or not.

N-50, q -0.3, p - 0.5, harshness - 0

```
mhd@pc:/media/mhd/New Volume/8th sem/CS765/Assignments/Assignment3$ python3 main.py --N 50 --q 0.3 --p 0.5 --harshness 0
Simulating for : Total : 50 voters, 15  Malicious Voters, and, 17 Benign Voters with correct vote probability 0.9, and 18
Voter Reputation Size:  (1000, 50)
Average Reputation Scores
First 15 fraction of voters: 0.0009980039920159682
Next 17 fraction of voters 0.9: 0.8989667723376775
Last 18 fraction of voters 0.7: 0.7003215790640941
```

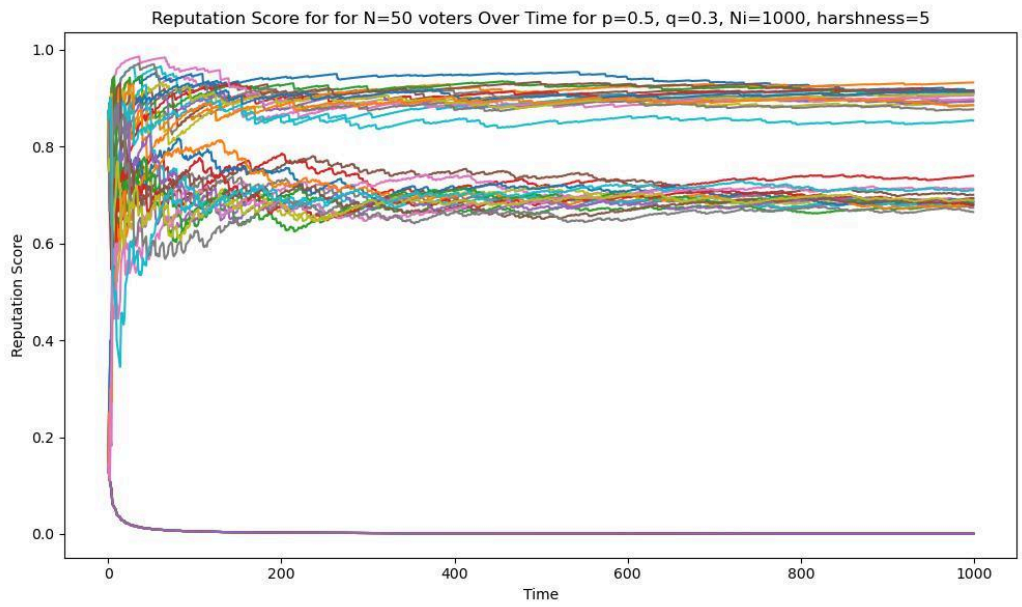
Since q is just 0.3, we see that the average reputation score of the malicious voters is close to 0 even with harshness = 0.



N-50, q - 0.3, p - 0.5, harshness - 5

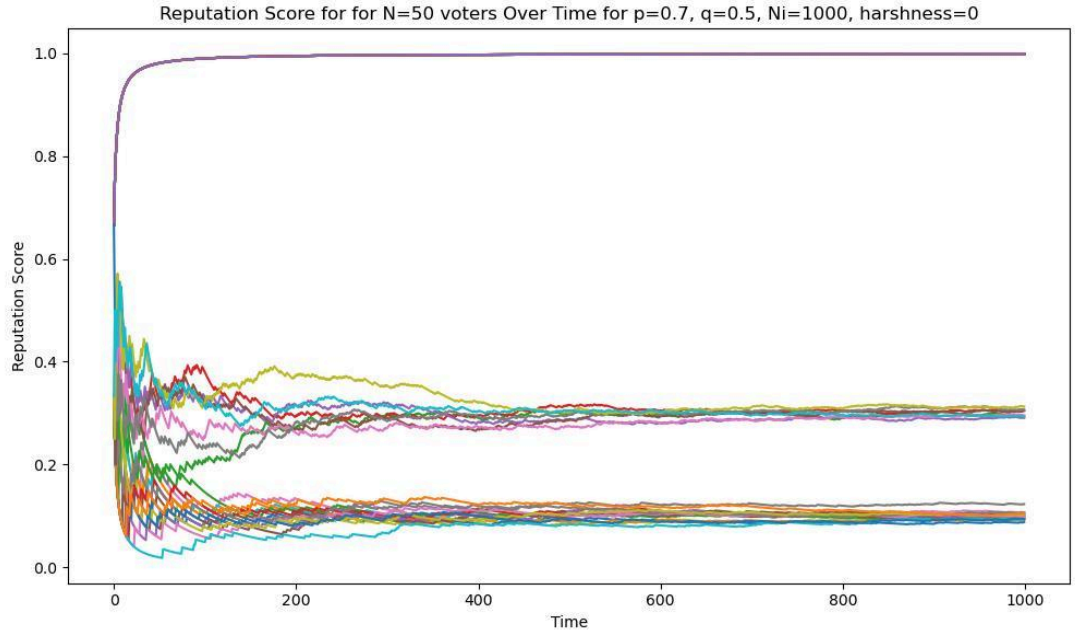
```
mhd@pc:/media/mhd/New Volume/8th sem/CS765/Assignments/Assignment3$ python3 main.py --N 50 --q 0.3 --p 0.5 --harshness 5
Simulating for : Total : 50 voters, 15  Malicious Voters, and, 17 Benign Voters with correct vote probability 0.9, and 18
Voter Reputation Size:  (1000, 50)
Average Reputation Scores
First 15 fraction of voters: 0.0005549389567147615
Next 17 fraction of voters 0.9: 0.9013514395769409
Last 18 fraction of voters 0.7: 0.6904982118633618
```

We observe that the reputation score of the malicious faction is converging to a nearer value to 0 by improving the harshness factor from 0 to 5.



N-50, q - 0.5, p- 0.7, harshness - 0

```
mhd@pc:/media/mhd/New Volume/8th sem/CS765/Assignments/Assignment3$ python3 main.py --N 50 --q 0.5 --p 0.7 --harshness 0
Simulating for : Total : 50 voters, 25  Malicious Voters, and, 17 Benign Voters with correct vote probability 0.9, and 8 B
Voter Reputation Size:  (1000, 50)
Average Reputation Scores
First 25 fraction of voters: 0.9990019960079838
Next 17 fraction of voters 0.9: 0.09974169308441938
Last 8 fraction of voters 0.7: 0.30014970059880236
```

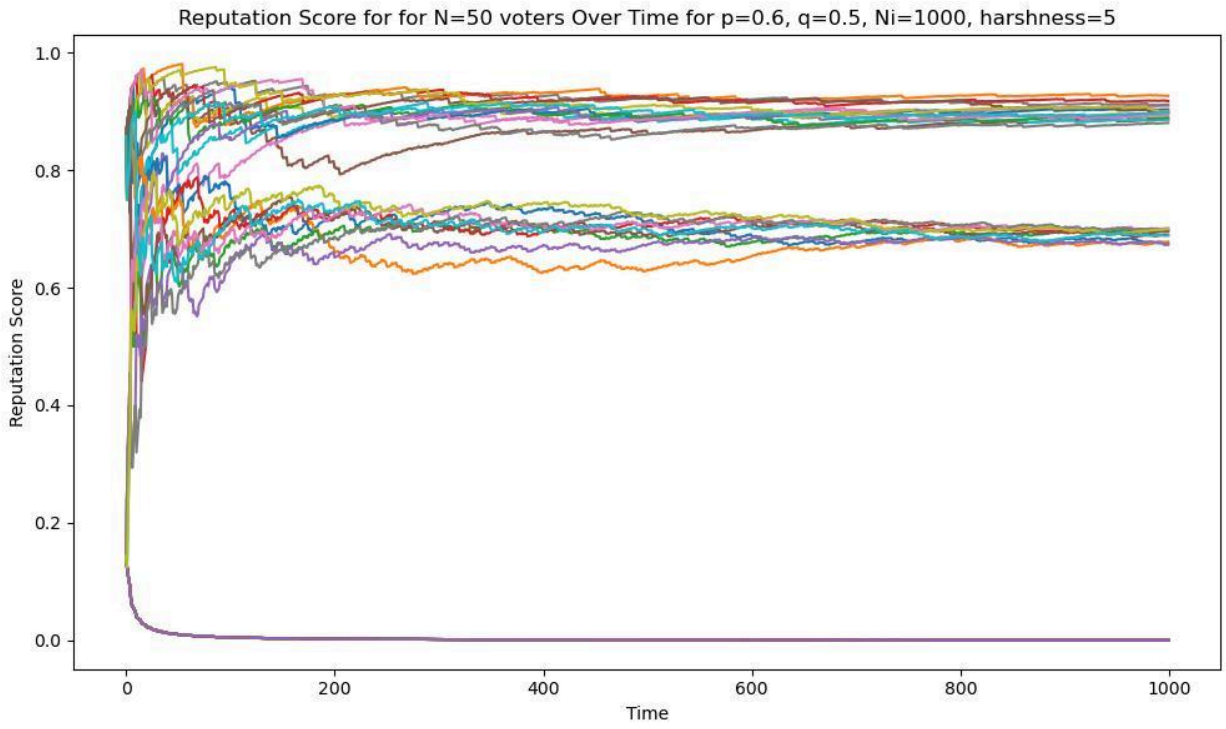


From our results above, we can see that since q = 0.5, malicious voters reputation score is converging to 1 as expected. We can also see that the fraction of honest voters who vote correctly with a probability of 0.7 have

their reputation scores getting converged to at a higher value than the honest voters who vote correctly with a probability of 0.9 because 0.7 fraction voters have a 0.3 probability to vote the same as the q fraction of malicious voters and thus their reputation score is converging to 0.3, whereas the other faction of voters are getting their reputation score converged to 0.1.

N-50, q - 0.5, p- 0.6, harshness - 5

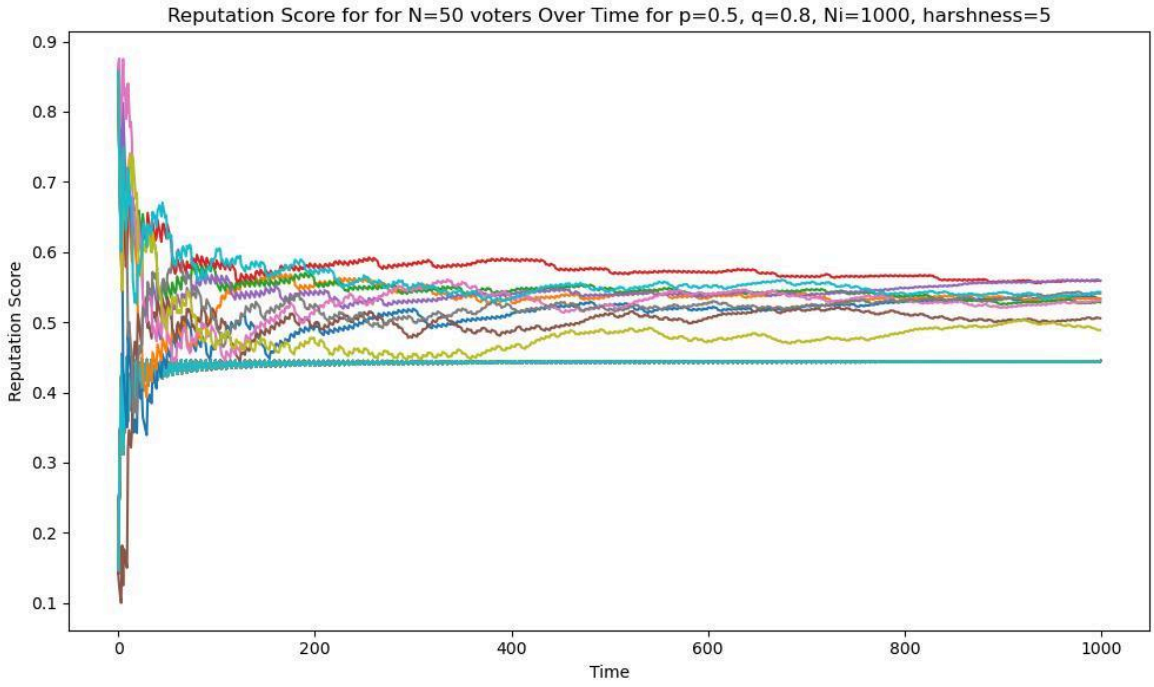
```
mhd@pc:/media/mhd/New Volume/8th sem/CS765/Assignments/Assignment3$ python3 main.py --N 50 --q 0.5 --p 0.6 --harshness 5
Simulating for : Total : 50 voters, 25  Malicious Voters, and, 15 Benign Voters with correct vote probability 0.9, and 10
Voter Reputation Size:  (1000, 50)
Average Reputation Scores
First 25 fraction of voters: 0.0005549389567147613
Next 15 fraction of voters 0.9: 0.8994080651128377
Last 10 fraction of voters 0.7: 0.6897891231964485
```



We see that our harshness parameter has worked as expected and it is confirmed by having the reputations of honest(0.9) ,honest(0.7), and malicious voters to converge to 0.9,0.7,0 respectively.

N-50, q - 0.8, p- 0.5, harshness - 5

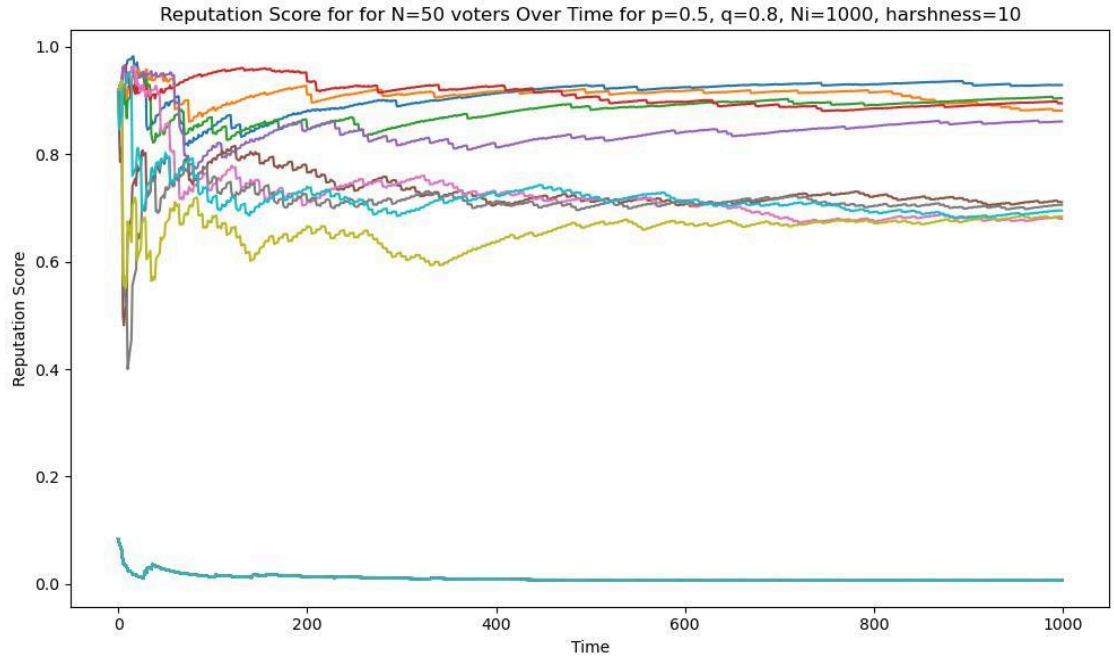
```
mhd@pc:/media/mhd/New Volume/8th sem/CS765/Assignments/Assignment3$ python3 main.py --N 50 --q 0.8 --p 0.5 --harshness 5
Simulating for : Total : 50 voters, 40  Malicious Voters, and, 4 Benign Voters with correct vote probability 0.9, and 6 Be
Voter Reputation Size:  (1000, 50)
Average Reputation Scores
First 40 fraction of voters: 0.4445061043285238
Next 4 fraction of voters 0.9: 0.5384295227524973
Last 6 fraction of voters 0.7: 0.5280244173140954
```



We get unexpected behavior here because the sheer quantity of malicious voters has increased, that our harshness of 5 is not having expected effect on our results. Lets try and increase it to 10 now.

N-50, q - 0.8, p- 0.5, harshness - 10

```
mhd@pc:/media/mhd/New Volume/8th sem/CS765/Assignments/Assignment3$ python3 main.py --N 50 --q 0.8 --p 0.5 --harshness 10
Simulating for : Total : 50 voters, 40  Malicious Voters, and, 4 Benign Voters with correct vote probability 0.9, and 6 Be
Voter Reputation Size:  (1000, 50)
Average Reputation Scores
First 40 fraction of voters: 0.00713775874375446
Next 4 fraction of voters 0.9: 0.9021234832262669
Last 6 fraction of voters 0.7: 0.7225791101594098
```



We can see that by adjusting our harshness parameter as q increases, we can make our reputation scores converge as expected even for higher values of q (0.8 to 0.9).