



# Intelligent Distributed Systems: Solving the Byzantine Generals Problem when Communication is Noisy

by

Santhakumar Chanrasekaram, Mats Oustad

Supervisor: Ole-Christoffer Granmo

**Project report for IKT 404 Distributed Systems in Spring 2007**

based on report template version 3.0 (2006)

Agder University College  
Faculty of Engineering and Science

Grimstad, 14 May 2007

Status: Final

**Keywords:** Byzantine generals, Tsetlin, Learning Automata, Noise, Distributed System

## **Abstract:**

*This project is built upon a previous exercise about solving Byzantine Generals problem. Now we will focus on solving the Byzantine Generals problem when the communication channel is noisy. To solve this we used a learning automata algorithm called Tsetlin automaton. The solution will work as long as the noise was within reasonable levels. The point of failure was the communication between commander and generals. If there was too much noise here, the outcome of the situation will be unstable.*

*This solution gives a way to eliminate noisy communication channel within reasonable levels.*

[This work is licensed under the Creative Commons Attribution-ShareAlike License \(http://creativecommons.org/licenses/by-sa/2.5/\).](http://creativecommons.org/licenses/by-sa/2.5/)

## Version

## Control

<b>Version<sup>1</sup></b>	<b>Status<sup>2</sup></b>	<b>Date<sup>3</sup></b>	<b>Change<sup>4</sup></b>	<b>Author<sup>5</sup></b>
1.0	Draft	07.05.07	Problem description	Santhan
1.1	Draft	09.05.07	Background	Mats, Santhan
1.2	Draft	10.05.07	Introduction	Santhan
1.3	Draft	11.05.07	Solution	Mats
1.4	Draft	14.05.07	Discussion	Mats
1.5	Draft	14.05.07	Conclusion and abstract	Santhan
1.6	Draft	14.05.07	Polished the document	Mats, Santhan
1.7	Final	14.05.07	Finished	Mats, Santhan

---

1 **Version** indicates the version number starting at 0.1 for the first draft and 1.0 for the first review version.

2 **Status** is DRAFT, REVIEW or FINAL

3 **Date** is given in ISO format: yyyy-mm-dd

4 **Change** describes the changes carried out since the previous version

5 **Author** is the one who did the change

# TABLE OF CONTENTS

<b>1 INTRODUCTION.....</b>	<b>4</b>
1.1 ACKNOWLEDGEMENTS.....	4
1.2 REPORT OUTLINE.....	4
<b>2 PROBLEM DESCRIPTION.....</b>	<b>5</b>
<b>3 BACKGROUND.....</b>	<b>6</b>
<b>4 SOLUTION.....</b>	<b>8</b>
4.1 REQUIREMENTS .....	8
4.1.1 <i>Functional Requirements</i> .....	8
4.1.2 <i>Non-Functional Requirements</i> .....	8
4.2 DESIGN SPECIFICATION .....	9
4.3 IMPLEMENTATION.....	12
4.4 VALIDATION AND TESTING.....	14
<b>5 DISCUSSION.....</b>	<b>17</b>
<b>6 CONCLUSION.....</b>	<b>18</b>
<b>APPENDICES.....</b>	<b>19</b>
REFERENCES.....	19

## 1 INTRODUCTION

This is a project in the course Distributed Systems at Agder University College. During this course we had some exercises with some known distributed systems and algorithms. We have done an exercise where we looked in to the Byzantine generals, made a small software program that demonstrated how the Byzantine worked. This project will be build up on that, with some new complications. The project is about implementing and testing a previously unpublished scheme that is able to solve the Byzantine Generals problem when the communication is noisy. This task was stated by our lecturer Ole-Christoffer Granmo.

To solve the task we will use the .NET environment, and C# as our programming language. We will also use web services, to represent commander and generals. We will add a noise filter to the communication channels, so that communication gets noisy. Then use a learning automata algorithm, Tsetlin automaton, to overcome the noisy channel to find out the real message. We will also test with different scale of noise to see if the outcome is the same as what commander intended.

### 1.1 ACKNOWLEDGEMENTS

We will thank Aleksander M. Stensby who was a former group member who helped us solve the exercise, which this project is built upon, the Byzantine Generals.

### 1.2 REPORT OUTLINE

The structure of this report is as follows. First we have problem description then followed by background, where we explain the main aspects of this project. Then we have the solution part, where we have the requirements, design specification, implementation and validation and testing. Afterwards we have discussion and conclusion followed by an appendix part where explanations of short terms and references could be found. In discussion chapter we will discuss the solution, if anything could have been done differently and so on.

## 2 PROBLEM DESCRIPTION

The goal of this project is to implement and test a novel and previously unpublished scheme that is able to solve the Byzantine Generals problem when the communication is noisy.

First thing we have to do is add noise to the communication channel. The noise in communication would appear between the commander and the generals, when the commander gives the order, retreat or attack, to the generals. The noise would also appear when the generals exchange information about the given order with each other; try to come to an agreement. Because of the noise in the communication channel a traitor could appear as a loyal commander and vice versa.

The noise in the communication channel is a big problem, since the outcome could be that everybody does the opposite of what the commander intended them to do. To overcome the noisy communication we would use a learning automata algorithm, Tsetlin automaton. We will test with different levels of noise.

Implementing, testing, and describing the proposed scheme will result in a successful project.

### 3 BACKGROUND

The aim of this project is to implement, test and describe the previously unpublished scheme that is able to solve the Byzantine Generals problem when the communication is noisy. Byzantine Generals problem is a well known problem, where we have a commander, some generals and traitors. Commander gives orders to the generals and they all contact each other to reach an agreement, to retreat or attack. The commander and each general will be represented by web services.

Tsetlin Automaton is a learning automata algorithm that is very simple and efficient to use in our case. This algorithm will help us to overcome the noisy communication, so that the generals take the action which was intended by the commander.

#### **Byzantine Generals** <sup>[2]</sup>

We can imagine that several divisions of the Byzantine army are camped outside an enemy city. Each division is directed by its own general and the generals, some of which are traitors, only communicate with each other by messenger. After observing the enemy, they must decide upon a common plan, to attack the enemy or retreat. Generals who are traitors will try to confuse and prevent the loyal generals from reaching agreement and make them adopt a bad plan.

##### ***Reaching agreement***

Reaching agreement could be a problem and Leslie Lamport came up with a solution [1] in 1982 for this problem. The solution was that the agreement could be achieved with  $k$  traitors with  $2k + 1$  processes. When the generals receive a message from the commander, they pass the message on to other generals. When generals receive message from either commander or a general, they store this message in their attack-list. When they have received attack or retreat message from all, processes make a decision. The type of messages received most is the decision.

E.g.: 3attacks, 2retreats => decision = attack

##### ***Detecting traitors***

Lamport came also up with a solution for detecting traitors. Detecting traitors can be detected with  $k$  traitors with  $3k+1$  processes. When the generals have received attack or retreat message from all, processes make a decision. The type of messages received most is the decision and the less received is the wrong message from the traitors.

E.g.: 5attacks, 2retreats => decision = attack (2traitors)

##### ***Noise on communication channel***

Noise can sometimes appear in a communication channel, which could cause problems for the byzantine generals. They can misunderstand the order from the commander or not get any order at all. The noise can appear between generals and could cause problems for them to come to an agreement.

## **Learning Automata**

The Learning Automata is a algorithm who adapts in order to enhance its performance on the task it is set to do. It learns by getting rewards when it does something right or gets punished if it does something wrong. This enables the algorithm to work in unknown or random environments, since it can adapt to make the decisions best for the state of the environment it is in. The ideas for Learning Automata were developed by Narendra and Thathachar. The different algorithms intuitively give them a lot of potential when it comes to solving problems that is difficult to find with conventional algorithms. An example is pattern recognition.

## **Tsetlin**

There are several different algorithms which are categorized as Learning Automatas, in this project we will be using the Tsetlin Automaton. Here one simply uses an integer variable which gets incremented for rewards and decremented for punishments. When the integer reaches a set number, positive or negative a final decision is made.

## **Web Service**

A Web Service is a way for computers on any platform to communicate in a safe and reliable way and make use of distributed transactions with a host over any form of TCP/IP network.

## 4 SOLUTION

### 4.1 REQUIREMENTS

---

#### 4.1.1 FUNCTIONAL REQUIREMENTS

##### A1: Byzantine Algorithm

Description: The solution needs a working implementation of the Byzantine algorithm

##### A2: Noise

Description: Noise in communication needs to be simulated

##### A3: Tsetlin Automaton

Description: Learning algorithm that is implemented to make sure that the noise does not affect the final decision of each general.

##### A4: Distribution

Description: The project needs to be written so that we can have connections between the different parts on different computers.

---

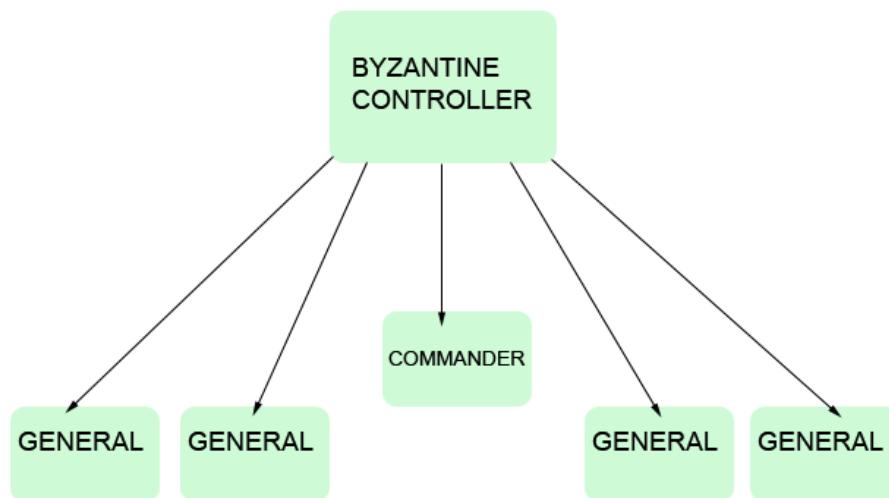
#### 4.1.2 NON-FUNCTIONAL REQUIREMENTS

Since this is a .NET project it need(s) a computer(s) with a Microsoft platform. The computers need to be connected to each other in some kind of network in order to obtain distribution.

## 4.2 DESIGN SPECIFICATION

For this project we are planning to use Microsoft Visual Studio 2005 for the development, it's designed for the .NET framework and therefore perfect for our choice of technology.

As a basis for our project we will use the design we made in Assignment three in this course.



*Figure 1: The architecture of Byzantine Controller*

As seen in the figure above there will be a Controller above the nodes in the Byzantine Algorithm, this is simply a GUI application to simplify the running and testing of the project, after the web services of the different nodes are made, they get added to the controller and we can link them, choose who will be traitor and then start the running of the algorithm through the controller.

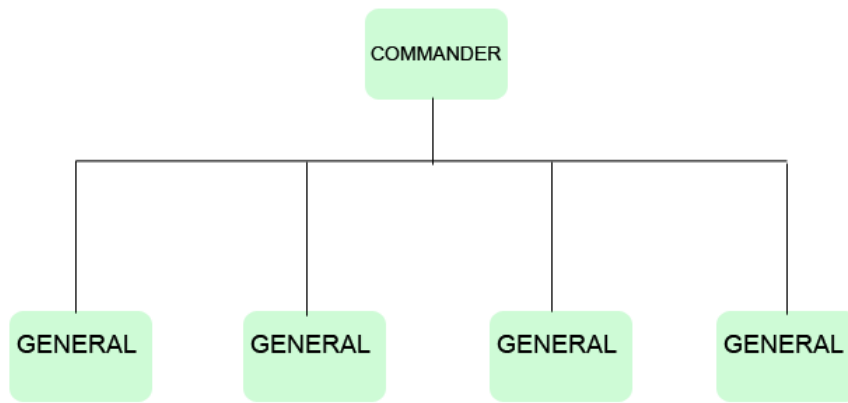


Figure 2: Byzantine commander and his generals

The commander issues the orders to each general, when the generals receive the order they start communicating to find out what the other generals have been ordered. There will be a chance of noise occurring in every step of this process, both when the commander issues orders and when the generals are communicating.

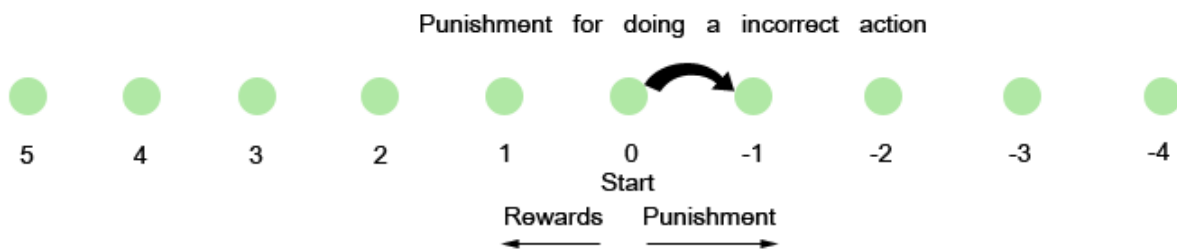


Figure 3: Tsetlin punishment



Figure 4: Tsetlin rewarding

That's why we will implement a Tsetlin Automaton, as shown in the figures this is an automaton which simply counts a integer value in positive or negative direction depending on the outcome. The goal of implementing this will be to overcome the problem the noise puts into the solution. Hopefully the generals will take the right final decision by learning through the automaton.

The Controller only gives the final decision of each general as feedback. In addition the solution will create text files which give a log of all the communication made between Command, Generals and Generals, Generals. This will hopefully make the testing and evaluation part simple and structured. We plan to run tests with different percentage

chance of noise occurring, from highly likely to happen to very rare, this will give a good view on how much it can damage the algorithm and of course also show if the Tsetlin Automaton is able to compensate for any severity of noise.

### 4.3 IMPLEMENTATION

As a basis for this project we used the code we produced in the Assignment three in this class. We've added functions, methods and variables needed to extend the code to match the problems and requirements.

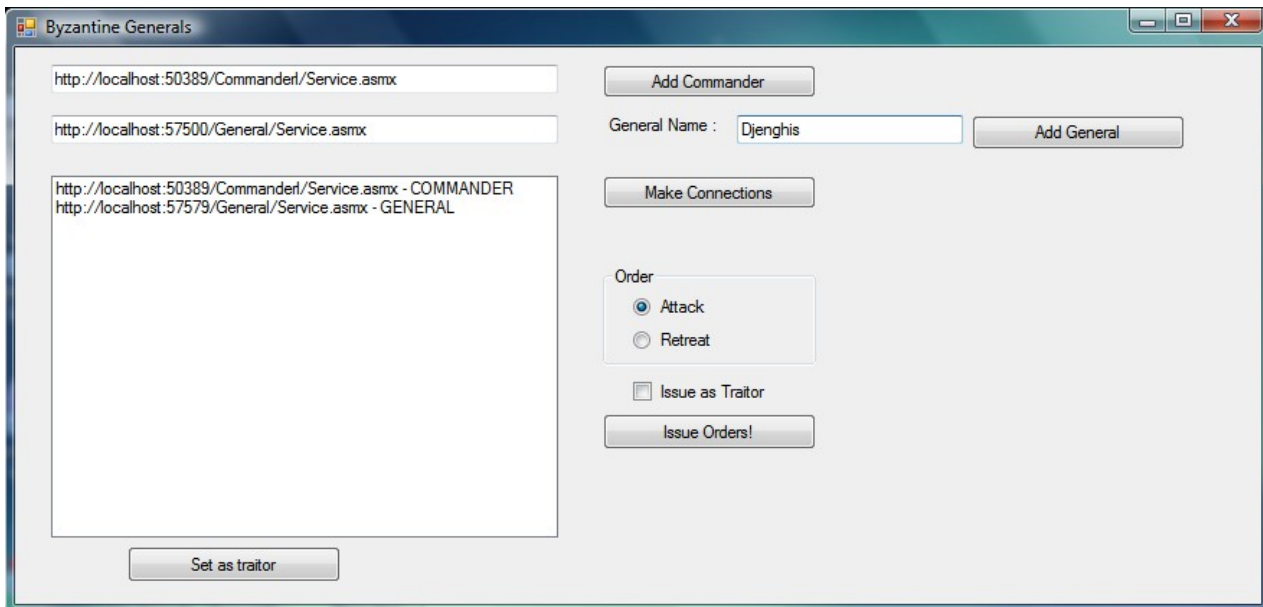


Figure 5: Simulation of Byzantine Generals Problem with noisy communication

In the controller commander and generals URLs are added. In addition each general also gets his unique name in order to separate them without separating on URLs. The make connections button connects all generals by sending a list of all the general URLs to each of them. In the list box you can mark a general and then pick him to be a traitor by pressing the Set as traitor button. One can choose the commanders order and if the commander is a traitor or not on the radio buttons and checkbox above the Issue orders button. This button is used to start the algorithm; the commander sends his order to each general and then the generals send messages between each other to determine what the final order will be.

```
public void tsetlin(int order)
{
    tsetlinnumber += order;
    s.Append("TSETLIN NUMBER ::: " + tsetlinnumber + "\r\n");
    if (tsetlinnumber >= 75)
    {
        finalOrder = 1;
        s.Append("FINAL ORDER =" + finalOrder.ToString() + "\r\n");
    }
}
```

```

else if (tsetlinnumber <= -75)
{
    finalOrder = 0;
    s.Append("FINAL ORDER =" + finalOrder.ToString() + "\r\n");
}
else
    finalOrder = -1;
}

```

*Function 1: Implementation of the Tsetlin Automaton*

The function shown above is the implementation of the Tsetlin Automaton. It simply takes an integer in as parameter and adds this to the Tsetlin number variable. The three inputs we send to this function is, +1, 0, -1 all depending on which conclusion the general makes for the current round. When the Tsetlin number reaches a given positive or negative number in this scenario shown we have used -75 and 75 the final order is set and the general has made up his mind. If the number hasn't reached these limits, the while loop around the getting orders from the other generals and finding out which order type has the most votes will keep running.

```

public int noise(int orderValue)
{
    int noiseOrder = orderValue;
    double numb = rand.NextDouble();
    if (numb > 0.94)
    {
        if (orderValue == 0)
            noiseOrder = 1;
        if (orderValue == 1)
            noiseOrder = 0;

        s.Append("NOISE OCCURED, NEW ORDER ::: " + noiseOrder.ToString() + "\r\n");
    }
    else
    {
        noiseOrder = orderValue;
    }

    return noiseOrder;
}

```

*Function 2: Simulation of noise*

This is the function we made to simulate noise in the communication. It uses a random generator which generates a number between 0.0 and 1.0, in case we have any number above 0.50 generates noise, this is of course a number we can change and is probably a bit high, but it is nice for testing purposes to get a lot noise cases. If noise occurs the order will change to the opposite of the original one. This gives a simple, but very effective way to create noise. The probability of noise occurring will be used to make different test results.

#### 4.4 VALIDATION AND TESTING

After some initial testes to see that everything was working as it should we decided to have eight test scenarios to test the algorithm.

TEST Number	No. of Generals	Noise Level	Traitor
1	3	40%	No
2	3	40%	Yes
3	3	20%	No
4	3	20%	Yes
5	3	10%	No
6	3	10%	Yes
7	3	5%	No
8	3	5%	Yes

We started of testing with high levels of noise and then slowly went down in order to see how that affected the running of the project and number of times it looped to reach the set Tsetlin number in either direction. We also did two test with each level of noise, one without any traitors and one where one of the generals were pre-set as a traitor. With noise both in communication between Commander, Generals and Generals, Generals, there are several places the noise can distort the decisions of the generals. With these test we see how good the Tsetlin automaton can compensate for this on the different levels of noise. All the tests conclusively show that we have fulfilled all the requirements and the solution is working as it should. To get test output we've used text files. The Commander and the Generals write out the data we need to analyse to it's own individual text file.

```
ORDER FROM COMMANDER : 1
::::::::::::GETTING ORDERS ROUND 0 ::::::::::::::
```

```
My ORDER :1
G1 ORDER :: 1
NOISE OCCURED, NEW ORDER ::: 0
G3 ORDER :: 0
Attackcount : 1
Retreatcount : 2
TSETLIN NUMBER ::: -1
```

```
::::::::::::DONE WITH ROUND 0 ::::::::::::::
```

Text above is gathered from General 2 in test 1 showing order it received from the Commander and the first round of running the loop. We know that the commander sent as order attack, but here we get noise from G1 and it seems G3 got the wrong order from the Commander, since this test was made without traitors. Retreat count is higher than the Attack count, which leads to Tsetlin number being decremented.

```
::::::::::::START TRANSFER::::::::::::
```

General G3  
ORDER SENT ::1  
NOISE IN COMMUNICATION!!!  
ACTUAL ORDER SENT :: 0

.....:END TRANSFER:.....

This is also extracted from test 1, and shows what the commander sent to General 3, here we can see he intended to send 1 which equals attack but noise occurred and the order got switched to retreat.

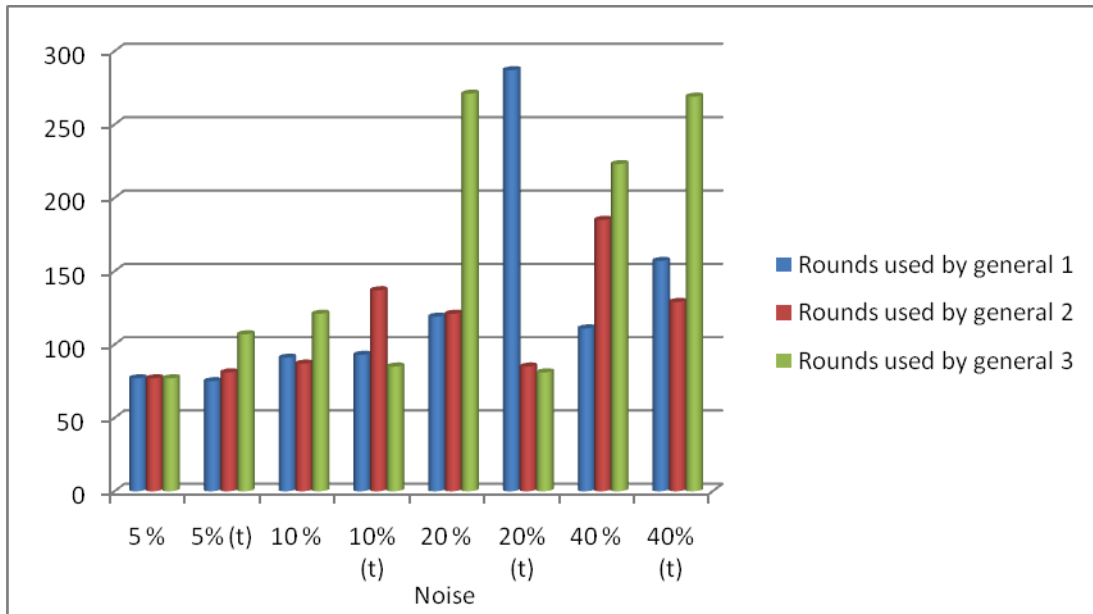


Figure 6: Test results 1

In the diagram above the results of each test are shown, they columns show how many rounds each general had to go through the loop before reaching the final decision. From the diagram we can see, higher noise leads to lengthier run-time.

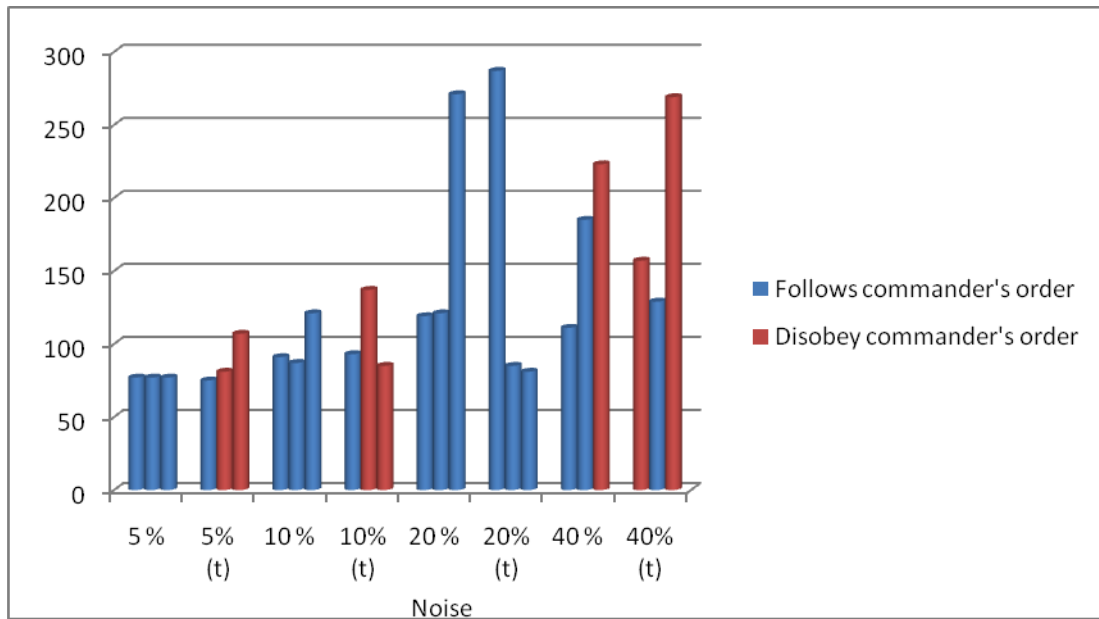


Figure 7: Test results 2

In the second diagram we have outline whether or not the generals reach as conclusion the order the commander sent. The blue columns indicate correct and the red indicates incorrect. There is a much higher number of blue columns in the no traitor tests than in the tests with traitor.

## 5 DISCUSSION

Our objective was to make a Byzantine simulation where there is noise in communication. Furthermore we were to implement a Tsetlin Automaton to see if this would compensate for the noise created. From our testing we have found out that a noisy Byzantine simulation will be unreliable, of course depending on how high the level of noise is. But we can also see that Tsetlin Automaton is doing a very good job compensating for the noise. In scenarios where there was no noise happening when the commander sent its orders, the Tsetlin compensated fully for the noise between the generals.

The Tsetlin is a simple algorithm, but it's very effective and intuitive when it comes to cases like this. If we take a closer look at test number 8, which only has 5 % noise but one traitor. Here we have the case that General 2 gets the wrong order from Commander because of noise, so he's told to retreat instead of Attack. General 1 is a traitor and will send retreat to the other generals even though he got attack from the Commander. This then leads to there being 2 retreats sent around instead of only Attacks and we then get wrong conclusion on General 2 and General 3. We have coded it so a traitor used the correct order when adding up his own list, that's the reason why General 1 draws the correct conclusion. We also see the same pattern in the other tests, In cases where there is one traitor and there is noise from Commander to any general not traitor, there will be generals coming to the wrong conclusion. But that is more related to the original demand of the Byzantine algorithm that there have to be  $2n+1$  generals where  $n$  equals the number of traitors. The noise from the Commander will in effect really create another traitor. If on the other hand the noise happens on a general who is pre-set as a traitor it will have the opposite effect. If we take the noise from Commander out of the equation we see that the Tsetlin is very effective, we have to go all the way up to 40% noise in order for it to fail and it's not very likely to have a network where it is 40% or above in noise levels. The speed of the solution is also were also good, with the highest level of noise we tested it took about one minute to finish the entire simulation, the test were done on one compute running all the web services.

The problem with the Commander to General noise could be solved by having the General ask for it's order for every iteration of the loop, then it will get the correct value in most of the rounds and the Tsetlin would also be able to compensate for this problem. This feature is something we in retrospect probably could have added in order to make the test cases even more conclusive. And will be a good idea to include in future work.

The use of web services is a very simple way to achieve distribution, and it also worked well for this project. Only objection was maybe the testing process took a bit more time than expected, this could probably have been shortened by developing the Controller GUI application some more, but since that wasn't really a part of the project, just a tool to help run things, it's wasn't prioritized. From working with this project we have learned about Automata and especially Tsetlin. How simple but powerful it can be.

## 6 CONCLUSION

The goal of this project was to implement and test a novel and previously unpublished scheme that is able to solve the Byzantine Generals problem when the communication is noisy. To solve this we added noise to Byzantine Generals problem simulation. The noise in communication would appear between the commander and the generals, when the commander gives the order, retreat or attack. The noise would also appear when the generals exchange information about the given order with each other. To overcome noise we used Tsetlin automaton, which is a learning automata algorithm.

This way of overcoming noisy communication channel worked very well as long as the noise was within reasonable levels. The point of failure was the communication between commander and generals. If there was too much noise here, the outcome of the situation will be unstable.

Adding Tsetlin automaton algorithm to the Byzantine Generals problem is a good way to prevent possible noisy communication, so that they don't interfere and influence the result. The problem with the Commander to General noise could be solved by having the General ask for his order for every iteration of the loop, then it will get the correct value in most of the rounds and the Tsetlin would also be able to compensate for this problem. This will be a good idea to include in future work.

## APPENDICES

## REFERENCES

Follow the IEEE guidelines for references

See: [http://www.ece.uiuc.edu/pubs/ref\\_guides/ieee.html](http://www.ece.uiuc.edu/pubs/ref_guides/ieee.html)

- [1] HiA, Bruk av kilder i skriftlige arbeider ved Høgskolen i Agder, august 2006, <http://www.hia.no/stud/eksam/kilder.htm>
- [2] The Byzantine Generals Problem, SRI International [online]. Available: <http://research.microsoft.com/users/lamport/pubs/byz.pdf>