1. Byzantine Agreement Problem

In the Byzantine agreement problem, n processors communicate with each other by sending messages over bidirectional links in order to reach an agreement on a binary value. Some processors are faulty who will try to confuse the others by sending different messages to different processors and prevent an agreement. The problem is to find an algorithm to ensure that the non-faulty processors will reach an agreement. The algorithm must guarantee that:

- All the non-faulty processors decide upon the same value.
- A small number of faulty processors can't cause the non-faulty processors to adopt a bad value.

The problem is not to find the faulty processors. It is the problem of reaching an agreement in a system where components can fail in an arbitrary manner and can send different messages to different components.

1.1 How Byzantine agreement problem works?

Every processor has some initial value and communication is done in rounds. Each processor sends its initial value to all the other processors. Let v_i be the message sent by the *ith* processor. Each processor then uses some method (say majority voting) to combine the values received $v_1, ..., v_n$ into single value, where n is the number of processors. A small number of faulty processors can affect the decision only if the correct processors were almost equally divided between two possibilities, in which case neither decision is bad.

For this approach to work every correct processor must obtain same values $v_1, ..., v_n$ so as to reach the same decision but a bad processor may send different values to different processors. This would mean that a processor cannot believe the value it gets directly from a processor i as it may be a bad processor. But we can't use a value different from the one sent by *ith* processor every time either, as the *ith* processor can be a correct processor. We need an additional condition to get rid of above possibility.

So for every i, the following conditions need to be met:
1. Any two correct processors use the same value of $v(i)$.
2. If the *ith* processor is correct, then the value that it sends must be used by every correct processor as the value of $v(i)$.

As we can see that we can restrict our consideration to the problem of how a single processor sends his

value to the others. This processor can be called "commander" and all the other processors are its "lieutenants". We can use the solution of this problem and get a solution to the original problem.

We can rephrase the problem as follows:
A commanding processor must send an order to his $n-1$ lieutenant processors such that
IC1. All correct lieutenants obey the same order.
IC2. If the commanding processor is correct, then every correct lieutenant obeys the order it sends.

1.2 Impossibility Results

If the processors can send only oral messages, then no solution works unless more than two-thirds of the processors are correct. Oral messages means the contents of the message are entirely in control of the sender. In particular, with only three processors no solution can work in the presence of single faulty processor.
Consider the scenario shown in the figure below:

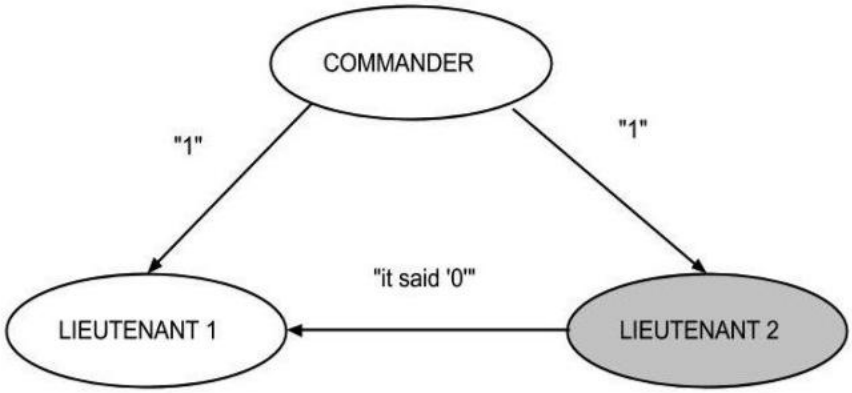

Figure 1(a): Lieutenant 2 is a faulty processor

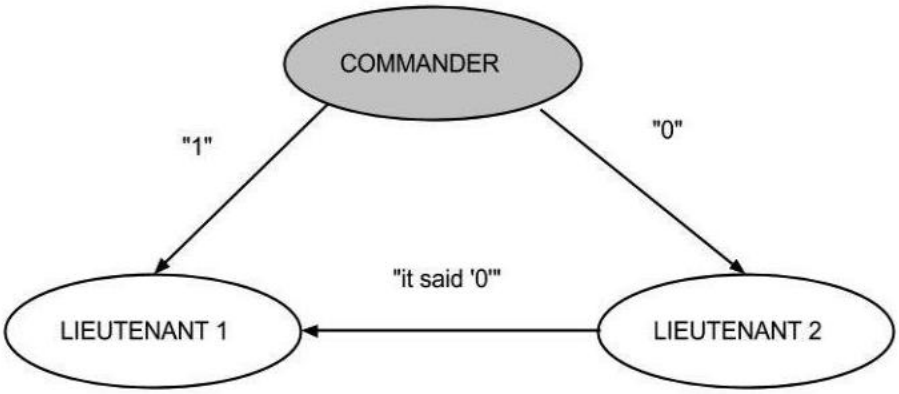

Figure 1(b): Commander is a faulty processor

In Figure 1(a), Commander sends a "1" to both the lieutenants but Lieutenant 2 sends message to Lieutenant 1 saying that it received a "0".
In Figure 1(b), the commander sends a "1" to Lieutenant 1 and a "0" to Lieutenant 2 and also Lieutenant 2 sends message to Lieutenant 1 saying it received a "0".

Lieutenant 1 cannot distinguish between both the situations if the faulty processor keeps lying consistently and wouldn't know which order to follow. Moreover, there is no central authority that everyone trusts which makes the problem hard. Hence, no solution exists for three processors in presence of a single faulty processor.

Using this result, it can be shown that no solution with fewer than $3m+1$ processors can cope with m faulty processors.

2. Solution: Oral Messages- No signature

The following assumptions are made of processors' message system:

A1. Every message that is sent is delivered correctly.
A2. The receiver of a message knows who sent it.
A3. The absence of a message can be detected.

A1 and A2 prevent faulty processor from interfering with the communication between other two processors. A3 will not let a faulty succeed who tries to prevent decision by not sending messages. Let's say default order is 0.

The algorithm uses function *majority* to combine the values received by each processor into single value and we have following choices for the value of $majority(v_1,...,v_{n-1})$:
- The majority value among the v_i if it exists, otherwise the value 0;
- The median of the v_i, assuming that they come from an ordered set.

Algorithm OM(0).
1.The commander sends his value to every lieutenant.
2.Each lieutenant uses the value it receives from the commander, or uses the value 0 if it receives no value.

Algorithm OM(m),m>0.

1.The commander sends his value to every lieutenant.
2.For each i, let v_i be the value Lieutenant i receives from the commander , or else be 0 if it receives no value. Lieutenant i acts as the commander in *Algorithm OM(m-1)* to send the value v_i to each of the other $n-2$ other lieutenants.
3.For each i, each $j \neq i$, let v_j be the value Lieutenant i received from Lieutenant j in step(2) (using *Algorithm OM(m-1)*), or else 0 if it received no such value. Lieutenant i uses the value $majority(v_1, ..., v_{n-1})$.

2.1 An example: Four processors and a single faulty (i.e. $m = 1$ and $n = 4$)

Case 1. One of the lieutenants is the faulty.
In step 1 of *OM(1)*, the commander sends the value "1" to all three lieutenants. In step 2, *OM(0)* executes and Lieutenant 1 sends the value "1" to Lieutenant 2 and Lieutenant 3. Similarly, Lieutenant 2 will send value "1" to Lieutenant 1 and Lieutenant 3. Lieutenant 3 is a faulty processor and it sends Lieutenant 1 and Lieutenant 2 different values: "1" and "0" respectively. In step 3, Lieutenant 1 makes decision $majority(1,1,1) = 1$ and Lieutenant 2 makes decision $majority(1,1,0) = 1$.

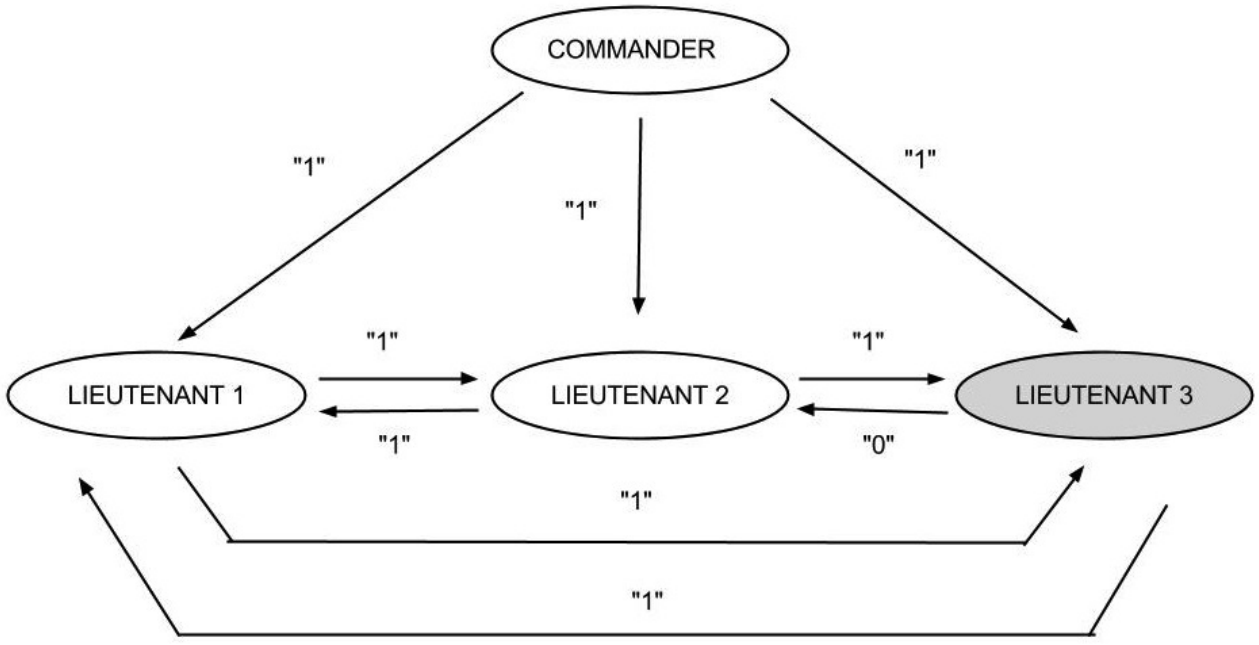

Figure 2: Messages sent and received among processors where Lieutenant 3 is a faulty processor.

Case 2. Commander is the faulty.
Commander is faulty and sends different values to three lieutenants as shown in the figure below. In step 3, Lieutenant 1 makes decision $majority(1,0,0) = 0$, Lieutenant 2 makes decision

$majority(0,1,0) = 0$ and Lieutenant 3 makes decision $majority(0,1,0) = 0$

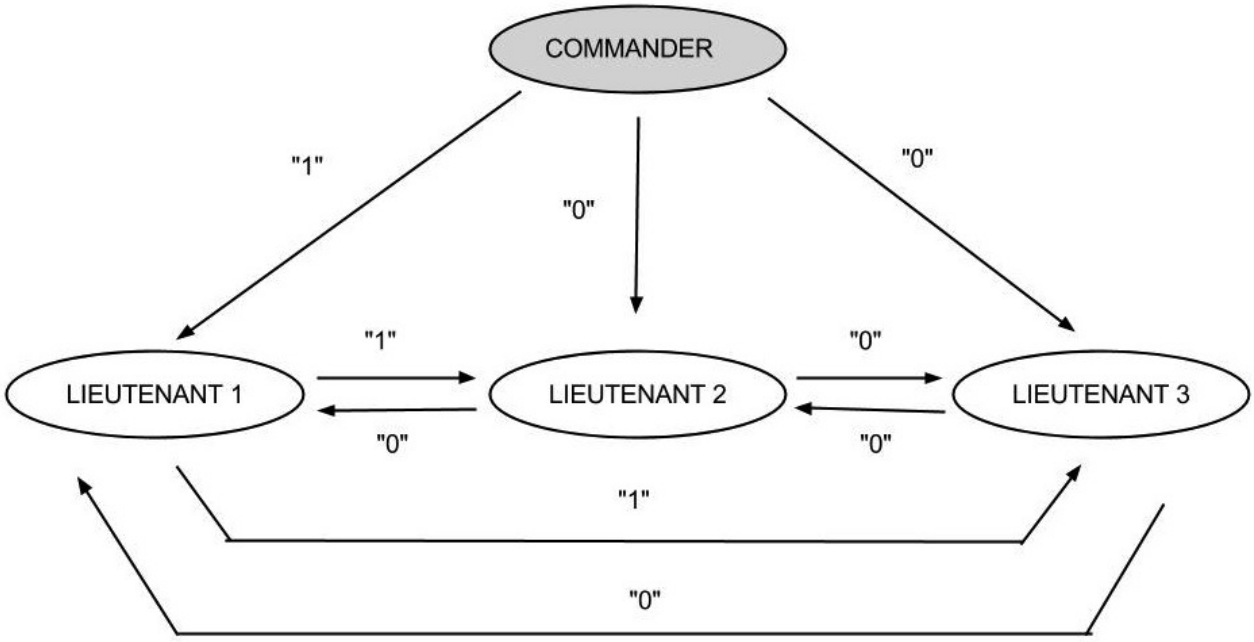

Figure 3: Messages sent and received among processors where Commander is a faulty processor.

2.2 Correctness of algorithm

Theorem 1. For any m, Algorithm OM(m) satisfies conditions IC1 and IC2 if there are more than 3m processors and at most m faultys.

Proof. This theorem can be proved by induction.
Base Case. For $m = 0$, there are no faulty processors. For *OM(0),* the commander sends its value to all the lieutenants (which are correct) and each lieutenant uses the value received from the commander. Each lieutenant will decide on the same value. Hence, *OM(0)* satisfies IC1 and IC2.
Inductive Hypothesis. Assume it is true for *OM(m-1).*
Prove it for *OM(m)*, $m > 0$.
Case 1. Commander is non-faulty.
For *OM(m),* the commander sends a value v to all the lieutenants. All the lieutenants receive the value v. But there are m faulty lieutenants, so each correct lieutenant can receive at most m faulty values. But we also know that the number of non-faulty processors is more than twice the faulty processors, so each correct lieutenant will receive majority of the $v_j = v$. Therefore, for each correct lieutenant $majority(v_1, ..., v_{n-1}) = v$. Hence, *OM(m)* satisfies IC1 and IC2.
Case 2. Commander is a faulty.

There are m faulty processors and the commander is one of them which means $m-1$ of the lieutenants are faulty. There are more than $3m-1$ lieutenants as there are more than $3m$ processors and $3m-1 > 3(m-1)$. By induction hypothesis, *OM(m-1)* satisfies conditions IC1 and IC2. Hence for each j any two non-faulty lieutenants get the same value v_j in step 3. Hence any two non-faulty lieutenants get the same vector of values and therefore the same value $majority(v_1, ..., v_{n-1})$ in step 3, proving IC1.

2.3 Number of messages

The agreement algorithms works by message passing among the processors. At each step of the algorithm, multiple messages can be sent. The time taken by the algorithm will depend on the time spent in exchanging messages which depends number of messages sent at each step of the algorithm.
The algorithm of Oral messages is recursive. For m faulty processors, there are $m+1$ rounds of message passing. The first round will have the format-"Here is my value". The second round will have format-"Processor p_1 said v" . The third round has format-" Processor p_1 said that processor p_2 said v" and so on.
OM(m) will invoke $n-1$ separate executions of *OM(m-1)* which in turn invokes $n-2$ executions of *OM(m-2)* and so on. The algorithm *OM(m-k)* will be called, $(n-1)...(n-k)$
There is a total of $(n-1)(n-2)...(n-m-1)$ messages exchanged.

3. Solution: Signed Messages

As we saw above the faulty processors can lie and we restrict this by sending unforgeable signed messages. We add the following assumption.
A4. (a) a non-faulty processor's signature can't be forged and any alteration of the contents of his signed messages can be detected.
(b) Anyone can verify the authenticity of a processor's signature.

A faulty processor's signature can be forged by another faulty, thereby permitting collusion among faulty processors.

This algorithm uses a function *choice* to reach final decision having following requirements:
1. If the set V consists of the single element v, then $choice(V) = v$.
2. $choice(\varphi) = 0$, where φ is the empty set.

$v:j:i$ denotes the value v signed by j and then the value $v:j$ signed by i. Let processor 0 be the commander. Each lieutenant i maintains a set V_i , containing the set of properly signed orders he has

received so far.

Algorithm SM(m).
Initially $V_i = \varphi$.
1.The commander signs and sends his value to every lieutenant.
2.For each i:
(A) If Lieutenant i receives a message of the form $v:0$ from the commander and he has not yet received any order, then
(i) he lets V_i equal $\{v\}$;
(ii) he sends the message $v:0:i$ to every other lieutenant.
(B) If Lieutenant i receives a message of the form $v:0:j_1:...:j_k$ and v is not in the set V_i, then
(i) he adds v to V_i;
(ii) if $k < m$, then he sends the message $v:0:j_1:...:j_k:i$ to every lieutenant other than $j_1,...,j_k$.
3.For each i: When Lieutenant i will receive no more messages, it obeys the order $choice(V_i)$.

3.1 An example
Commander sends a "1" to Lieutenant 1 and a "0" to Lieutenant 2. Both the lieutenants receive the two orders in step 2. After step 2, $V_1 = V_2 = \{"1","0"\}$ and they both obey the order $choice(\{"1","0"\})$. Here, the lieutenants know the commander is a faulty because his signature appears on two different orders and A4 states that only he could have generated those signatures.

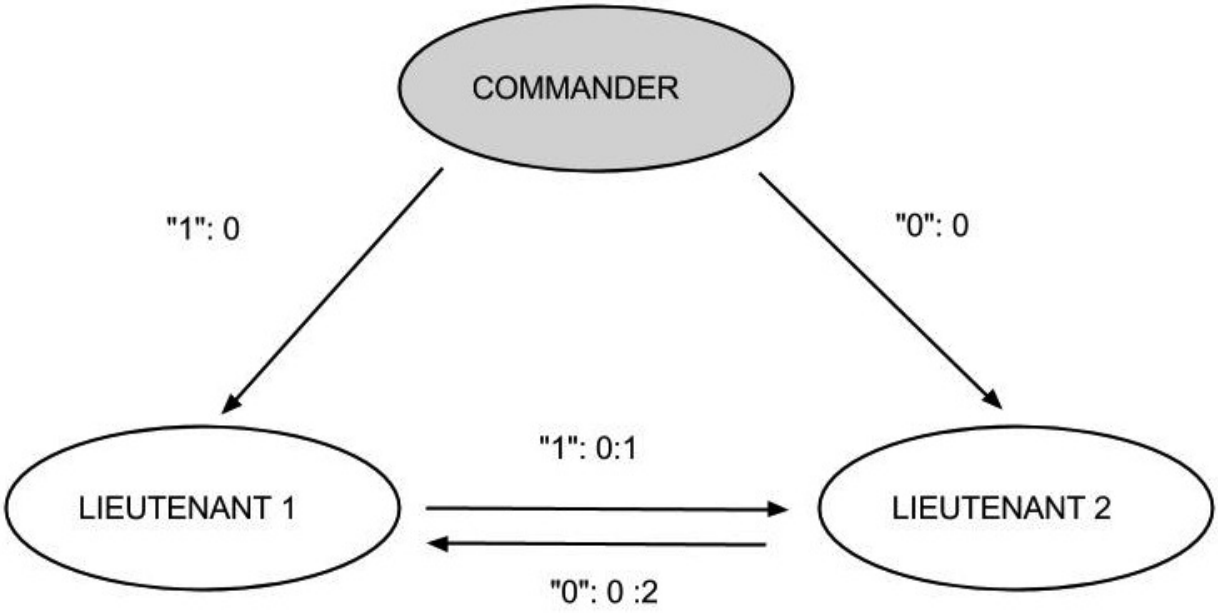

Figure 4: Commander is faulty

3.2 Correctness of algorithm

Theorem 2. For any m, Algorithm SM(m) solves the Byzantine Problem if there are at most m faulty processors.
Proof. We need to prove that algorithm always satisfies IC1 and IC2.
Case 1. Commander is non-faulty.
Step 1. Commander sends order $v : 0$ to every lieutenant.
Step 2(A). Every non-faulty lieutenant receives the order v.
Step 2(B). non-faulty lieutenant receives no more orders as the commander is non-faulty and there can be no more orders of the form $v' : 0$.
Step 3. As V_i contains only one order. Each non-faulty lieutenant i ends up obeying this order.
Hence, IC1 and IC2 are satisfied.

Case 2. Commander is faulty.
In this case we just need to prove that IC1 holds. To prove that IC1 holds we need to show that the set of orders V_i and V_j are same. For this we consider how two non-faulty lieutenants i and j send and receive orders. Situation 1 is where i receives an order $v : 0$ in step 2(A), adds it to V_i and then it sends it to j too by step 2(A)(ii). Situation 2 is where i receives the order $v : 0 : j_1 : ... : j_k$ in step

2(B). If j is one of $j_1 : ... : j_k$, then it must have received the order already otherwise there are two cases:

1. $k < m$. By 2(B)(ii), i sends the order $v : 0 : j_1 : ... : j_k : i$ to every lieutenant other than $j_1 : ... : j_k$. So, j receives it too.
2. $k = m$. The commander is faulty, that leaves us with $m - 1$ faulty lieutenants. As $k = m$, at least one of the lieutenants $j_1 : ... : j_k$ is non-faulty which means this non-faulty lieutenant must have sent j the order v as soon as it first received it.

Hence, the theorem is proved.

References

http://research.microsoft.com/en-us/um/people/lamport/pubs/byz.pdf