1 Introduction:

This lecture note presents the algorithm that solves the Byzantine Generals Problem, as first described by Lamport, Pease, and Shostak in 1982 [6]. The Byzantine Generals Problem is one of many in the field of agreement protocols. This problem is built around fictitious characters of Byzantine Army with each army division having its own General. The Generals are camped outside the enemy territory and they are geographically separated. The generals need to decide whether to attack or retreat by observing the enemy movements. All the generals need to reach a proper agreement for them to win the battle. The problem seems simple until we introduce traitor generals with arbitrary behavior (Note: The behavior of all the traitors can be assumed to be co-ordinated by a single adversary and there is no limit on the computational ability). The loyal generals will be destroyed in the battle if they do not all take the same action, so they need to reach some agreement but there are traitors amongst them who will try to conspire to prevent them from reaching a consensus. We will reduce this N generals problem to 3 generals problem to better understand the agreement problem.

1.1 3 Generals Problem:

The agreement problem can be clearly understood when we consider the reduced version of N-Generals Problem by considering the 3 Generals Problem as explained in [6] where there is a commanding general and 2 other lieutenant generals who have to obey his order when consensus is reached. To keep it simple, we consider the case where the battle plan is to either "attack" or "retreat". For the 3 generals problem we consider the existence of one traitor. There are two cases where one of the lieutenant is the traitor and in the other case commanding general himself is the traitor.

- Case 1: Consider Fig 1 in which the commander is loyal and sends an "attack" order, but Lieutenant 1 is a traitor and reports to Lieutenant 2 that he received a "retreat". Lieutenant 2 must obey the order to attack because he is being commanded and he is loyal.This leads to a scenario where the Lieutenant has to obey in presence of no consensus also.

- Case 2: Consider Fig 2 in which the commander is a traitor and sends an "attack" order to Lieutenant 2 and a "retreat" order to Lieutenant 1. So if Lieutenant 2 obeys commander he must attack and if Lieutenant 1 obeys commander he must retreat which introduces us to the agreement problem.This implies that lieutenants need to listen to each other to detect a traitorous commander.

We can see that for Lieutenant 2 receives the same value $\{Attack, Retreat\}$.You can see the difficulty Lieutenant 2 faces in this situation. Regardless of which situation he is in, the incoming data is the same. He has no way to distinguish between the two configurations, and no way to know which of the two generals to trust.

Hence this shows that there exists no solution for 3 Generals problem with 1 traitor.This shows that the traitor general can destroy the loyal generals by fooling the loyal generals into making a decision which they hadn't decided upon. Byzantine fault tolerance can be achieved, if the loyal (non-faulty) generals have a unanimous agreement on their strategy. Note that if the commander general is correct, all loyal generals must agree upon that value. Otherwise, the choice of strategy agreed upon is irrelevant. This is a common scenario in the field of distributed domains, for example, when it is necessary to agree whether to commit or abort a transaction

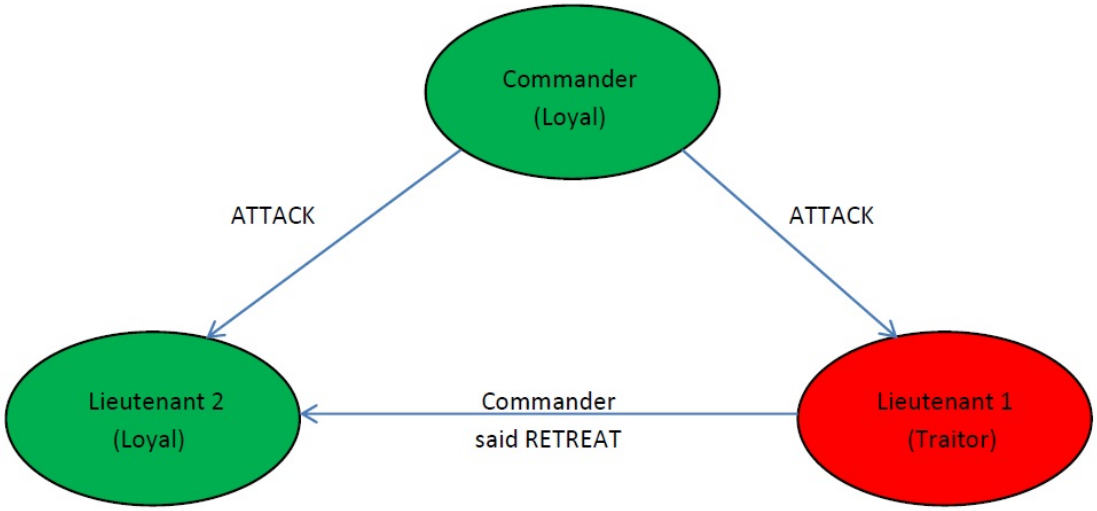

Figure 1: Lieutenant 1 is traitor

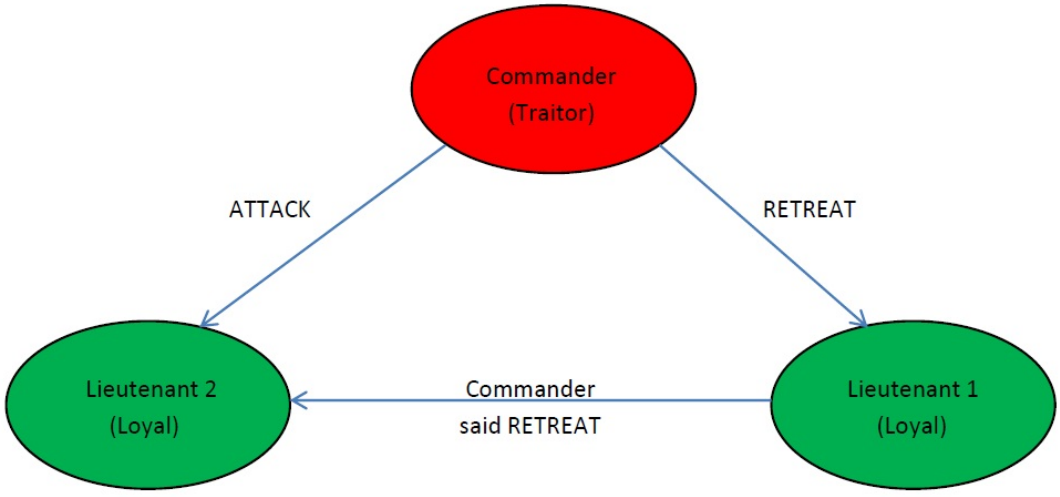

Figure 2: Commander is traitor

in a distributed database. We will discuss the motivation behind studying this in detail to get us a good insight into practical aspects of Byzantine agreement

2 Motivation:

It is said "To err is human, but to really foul up takes a computer". Computers are used where system failure would be catastrophic in terms of money, human lives, or ecosystem. Applications like Process Control, Patient Monitoring, Systems, Missile guidance and Control, Air Traffic Control, Fly-by-Wire Aircraft, Transaction Processing, Stock Market require that the computers are robust and the system does not fail.We know by definition of a distributed system there does not exist a centralized controller. Thus distributed solution methods requires that all the system reach some consensus. The distributed system can be characterized as

- Asynchronous : These systems do not make any assumption about the timing and there exists no time outs.
- Synchronous: These systems are the ones which permits time outs.

A Formal model of a distributed system can be modeled as a Graph $G = (V, E)$

- $\mathsf{V} = N$, i.e. there are N nodes.
- $\mathsf{E} \leq \frac{N^2-N}{2}$, where E is the number of communication channels(links).

Failures can cause a machine to give the wrong result for some inputs. Some of the types of failures in distributed systems are Persistent or Intermittent, Node Failure vs Communication Failure, Security intrusions can be modeled as failures.

The main motivation of discussing this problem is to construct a Fault tolerant distributed system. To construct it we need to understand the kinds of faults that exist and the system should continue to operate properly in the presence of a reasonable number of failures. The failures can be

- Fail Stop: These kind of failures are the ones where nodes/links shut down .(Power failures). They are simple and non-malicious errors. Processor halts instead of performing erroneous transformations.
- Byzantine: The failed nodes/links give incorrect values. These faulty nodes can crash deviate from the protocol,crash, act selfishly, lie, equivocate, attack other nodes. These are the kind of errors that are malicious and can be assumed to be generated by a computationally powerful adversary.

In distributed system terminology, the generals are collectively known as processes,one general has previously been elected to start the process. Traitorous generals and lieutenants are faulty processes, and loyal generals and lieutenants are correct processes. The order to retreat or attack is a message with a single bit of information: a one or a zero. The general who has been elected to start may be a traitor and the loyal generals need to come to an agreement to retreat if this is true.

In general, a solution to an agreement problem must pass three tests: termination, agreement, and validity. As applied to the Byzantine Generals problem, these three conditions are:

- A solution has to guarantee that all correct processes eventually reach a decision regarding the value of the order they have been given.

- All correct processes have to decide on the same value of the order they have been given.
- If the source process is a correct process, all processes have to decide on the value that was original given by the source process.

If these tests are satisfied, it is easy to see that consensus can be achieved on a single bit. A consensus reached on a single bit is a satisfactory condition for consensus to be reached for multi-bits. This follows from the fact that if consensus on multi-bit is required, we could run the algorithm on every bit and arrive at a consensus.

3 Byzantine Generals Problem:

We previously discussed about the problem and now we will try to reason out under what conditions can all loyal generals co-ordinate their action so as to either attack in unison or retreat. There are few assumptions we need to consider before we discuss any further about the requirements and they are

- Generals start the protocol with individual opinions on the best strategy *Attack* or *Retreat*.
- All loyal generals do what the algorithm asks them to do. This means that loyal generals are deterministic and at any point we know what value each one is going to send.
- Traitor generals sends random values to other generals and can lie about the state he or she holds.

The next subsection describes the requirements for protocols which solve the consensus problem generally referred to as "Byzantine Agreement" protocols, to be robust.

3.1 Requirements:

All robust algorithms are centered around the solution of decision problems and in our case it is required that each of the loyal generals irreversibly makes a decision to either *Attack* or *Retreat*

Any Byzantine Agreement protocol must pass these three conditions:

- Validity
- Agreement
- Termination

We will try formalize these conditions for our protocol by discussing about the possible solutions and reason our way to some general conditions known as Interactive consistency .

In deterministic algorithms, termination is required in all possible computations. With regard to our problem at hand the *Termination* can be stated as "All loyal generals agree upon some decision eventually."

Naïve Solution: Consider a case where there are no traitors. This problem reduces to all the generals sending their local information to every other general and agreeing upon common battle plan. They can agree upon the value of attack or retreat based on some majority voting scheme once they receive all the messages. This solution works as long as there are no traitors. But with the introduction of small number of traitors, what we have is that, the traitor now can send attack to half of the loyal generals and retreat to the other half. This confuses the

loyal generals into making decisions of attack and retreat equally similar to tossing of a coin as seen in Fig 1. So now we face a dilemma of how to describe a bad decision as both possibilities are equally possible.
Thats why this is a naïve approach and it assumes a method by which the generals broadcast their values v to one another. However, majority voting method does not work, because this requires that every loyal general obtain the same values $< v(1).....v(n) >$, and a traitorous general is controlled by an adversary who makes the traitorous general to lie consistently.
This leads to an important requirement for our protocol which is the *Validity* which can be stated as "All loyal generals have the same value at the end".
Thus we see that every value $v(i)$ sent from the i^{th} general must be verified by all the loyal generals before agreeing on that value. This is nothing but our *Agreement* condition which can be re-stated as "All loyal generals decide on the same value $v(i)$."

Reduction of N-Generals Problem: In the previous section we saw that the naïve approach failed because of the difficulty to find a robust method to combine all the values received which satisfies the *Validity* and *Agreement* conditions. Also at the end of the naïve solution we saw that loyal generals had to have the same value $v(i)$ for any given i . Using this we can refine our approach by reducing the N-Generals problem to a problem where there is a randomly chosen leader called Commander general who sends out his values to all other subordinate generals called the Lieutenant generals and the Lieutenant generals discuss amongst themselves what the Commander's order was. So the conditions for the protocol to be robust can be stated as Interactive consistency conditions. This is discussed in the next subsection.

3.2 Essential Conditions:

There exists a solution for the situation with a commanding general sending an order to his $n-1$ lieutenant generals if it satisfies the below conditions

- Validity: All loyal lieutenants obey the same order.
- Agreement: If the commanding general is loyal, then every loyal lieutenant obeys the order he sends

The above statements describe the agreement requirements. Conditions Validity and Agreement are called as the *interactive consistency* conditions in distributed system terminology per se. Also we can see if the Agreement is true then we have that all loyal lieutenants obey the same order which was send from commander and hence it implies Validity.
Till now we discussed the essential conditions for a protocol to be robust but we will now see that the protocol even though robust cannot solve the agreement problem under certain simple conditions and thus the difficulty of this problem is exposed to us.

4 Impossibility Results:

We discussed that there does not exist a solution for 3 Generals Problem where 1 was a traitor. The same intuition will be used to reason out that, there exists no solution for $3m$ generals with at-least m traitors. We will try to prove by contradiction

Lemma : There exists no solution for a group of $3m$ or fewer generals with m traitors.

Solution: To prove this we will consider there exists a solution to our at-most $3m$ generals with m traitors. So now we can partition these at-most $3m$ generals into three groups A,

B, C such that each group has at-least 1 and at-most m generals. Now we will consider randomly one general out of the three groups to be our Commanding general. So say group A has Commanding general, then we have group A has at-most $m-1$ Lieutenants. Also say a group B is traitor general's group. So we have that at-most m traitors are present. So we have that if commander was loyal all the other lieutenant generals in group A follow his order as all are loyal and hence Agreement condition holds good and and also this implies Validity condition. If the Commander was traitor then the in group A all the lieutenants must be traitors. This suggests that those $m-1$ lieutenants can do anything they want and hence group B and group C decide upon retreating which gives the Agreement condition and Validity condition. So we have now reduced $3m$ Generals Problem to 3 Generals Problem and we have proved that there exists a solution which contradicts our Lemma and 3 Generals reasoning and hence there exists no solution for $3m$ or fewer generals with m traitors.
So using this we can say that there exists a solution only if essential conditions are met and there exists more than $\frac{2}{3}^{rd}$ of the Generals are loyal.

5 Oral Message Algorithm:

We will discuss an algorithm that tries to solve Byzantine agreement problem in this section. We already know that when the generals communicate amongst themselves through oral messages, then impossibility solution gives the bound on number of traitor generals to be less than $\frac{1}{3}^{rd}$ of the total. To give a brief insight into the algorithm, the commanding general starts the protocol with individual opinion on the best strategy: to attack or to retreat. The lieutenant generals exchange oral messages to execute the protocol, and if they decide on attack in some round say i then they attack in the $i+1$ round. If there is no agreement in the i^{th} round then every general broadcasts the value he has received in the previous round to all other remaining generals (generals who have not yet become the commander). So this repeats for $m+1$ rounds where m is the number of traitors. The final goal is to arrive at a state in which, no matter what the traitor generals do, the loyal generals are able to force a unique outcome all by themselves.

Now that we discussed about Oral message, let us try to analyze more on what this oral message is. Oral message is nothing but an unauthenticated message. So an oral message is a message which the traitor can forge the message and claim that it was received from another general and the traitor can tamper with the contents of a received message before relaying it. So we have to make clear assumptions on certain conditions which must be satisfied for all the messages for this protocol to work. Point to point communication of the messages happen and as it is point to point every receiver knows the sender and also can detect the absence of message. These assumptions are required to place certain restrictions on the adversary. The point to point communication makes sure that the adversary cannot interfere with messages sent between two generals, and adversary cannot introduce some spurious message as every sender can be tracked back. Also we are assuming absence of messages which limits the adversary from not sending a message.

5.1 Algorithm:

We present the pseudo code for the algorithm below.

Initial Conditions:

- v = Initial value.
- m = # of traitors.
- n = Total # of generals.
- L_0 = Initial commander.
- Start $OM(m, 0, v)$ with 0^{th} general as the commander.

Algorithm OM($\mathbf{m}, \mathbf{t}, \mathbf{v}$).

- Commander = L_t initiates oral byzantine protocol.
- *send(v)* to $n - t$ lieutenant (L).
- for ($i = t$ to $n - 1$):
  v_i = *receive(v)* from C by L_i.
  $OM(m - 1, i, v_i)$.
- for ($i = 1$ to $n - 1$):
  v_j = *receive(v)* from L_j by L_i.
  final v_i = *majority* ($< v_0, v_1, ...v_{n-1} >$).

Here we assume three functions *send(v), receive(v), majority*($< v_0, v_1, ..., v_{n-1} >$). The send function is executed by a commander general who sends the value v to the lieutenants and the receive function is executed by the lieutenant generals and they receive a value v from a commander. If by chance the value v is lost then we use a default order and we assume it to be Retreat for our algorithm. Also the $OM()$ which is our main procedure takes the parameters of m which is number of traitor, t which represents the lieutenant (l_t) who acts as a commander in that round and v the value he sends.

5.2 Working of Oral Messaging Solution:

1. **Example :** Consider an example where we have $m = 1$ and $n = 4$ with Lieutenant 3 being a traitor. Figure 3 shows the $OM(1, 0, A)$ execution and the messages received by all the lieutenants. Initially the commander sends the same value *"A"* to all the lieutenants (because the commander is loyal).

   Figure 4 shows the $OM(0, t, v)$ execution where every lieutenant t now acts as a commander and forwards the value he has received, from the actual commander, to all other lieutenants. Lieutenants 1 and 2 being loyal in this case, forward the value *"A"* which they received in $OM(1, 0, A)$ without tampering with it. But since lieutenant 3 is a traitor, he changes the value *"A"* to *"R"* and sends that to lieutenant 2 and 3.

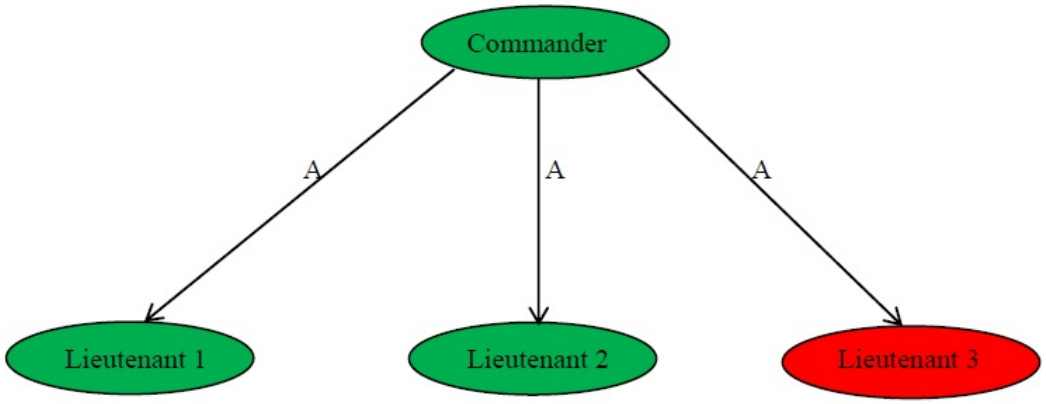

Figure 3: OM(1,0,A): Lieutenant 3 is a traitor

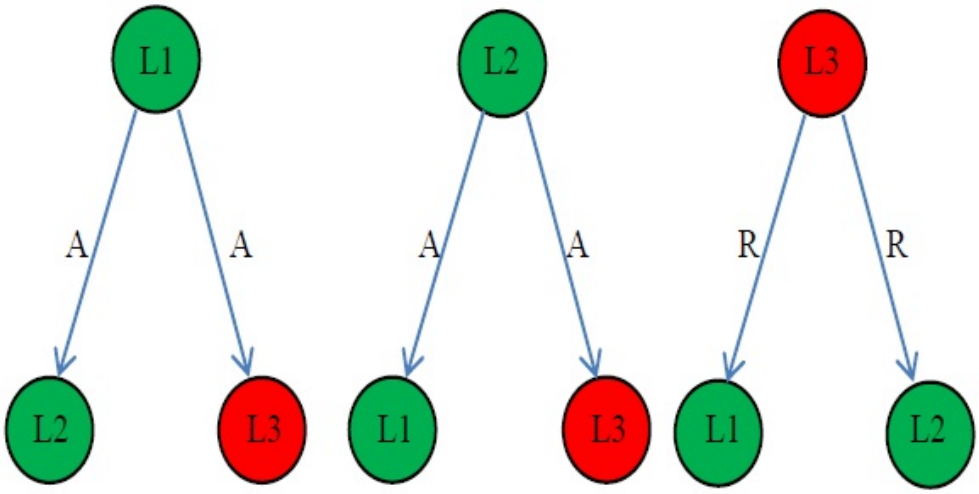

Figure 4: OM(0,t,v) state

After the message exchanging is completed, the 3 lieutenants will have the following values in their respective sets:

- L1 = $\{A, A, R\}$
- L2 = $\{A, A, R\}$
- L3 = $\{A, A, A\}$

After applying the majority function to these sets, the lieutenants will end up with the following values: L1 = A, L2 = A, L3 = A. Hence 1 traitor general was handled by 3 loyal generals.

2. **Example :** Consider the same example where we have $m = 1$ and $n = 4$, but with commander being a traitor. Figure 5 shows the execution of $OM(1, 0, v)$ in this case. Assume that the traitorous commander can now send any arbitrary values apart from attack and retreat. For instance, here he sends X, Y, and Z to the three lieutenants.

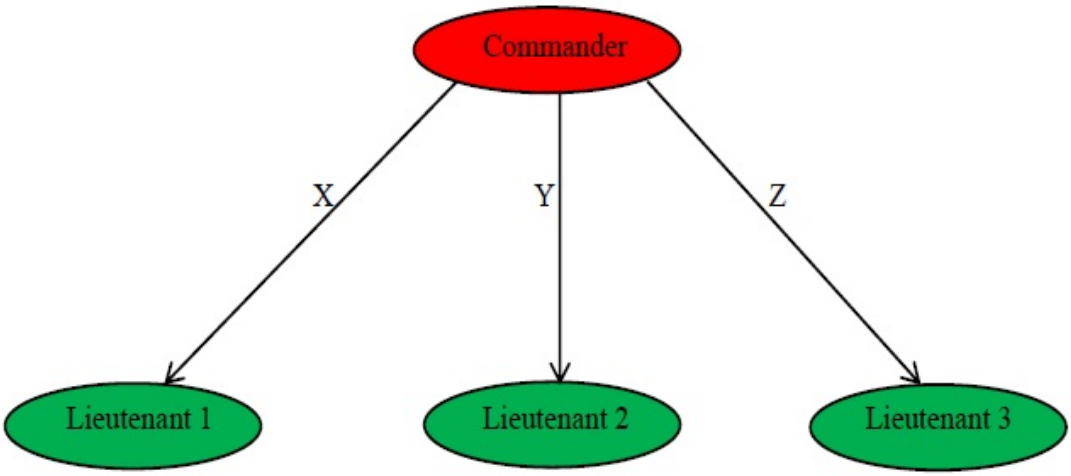

Figure 5: OM(1,0,v): Commander is a traitor

Figure 6 shows the $OM(0, t, v)$ execution where every lieutenant t now acts as a commander and forwards the value he has received, from the actual commander, to all other Lieutenants. In this case all the lieutenants are loyal so they simply forward the values X, Y and Z to each other.

After the message exchanging is completed, the 3 lieutenants will have the following values in their respective sets:

- L1 = $\{X, Y, Z\}$
- L2 = $\{X, Y, Z\}$
- L3 = $\{X, Y, Z\}$

All the lieutenants end up with the same set of values. Since there is no single value existing in majority in these sets, the lieutenants follow the default value. Hence, L1 = L2 = L3 = default value. Hence 1 traitor general was again handled by 3 loyal generals.

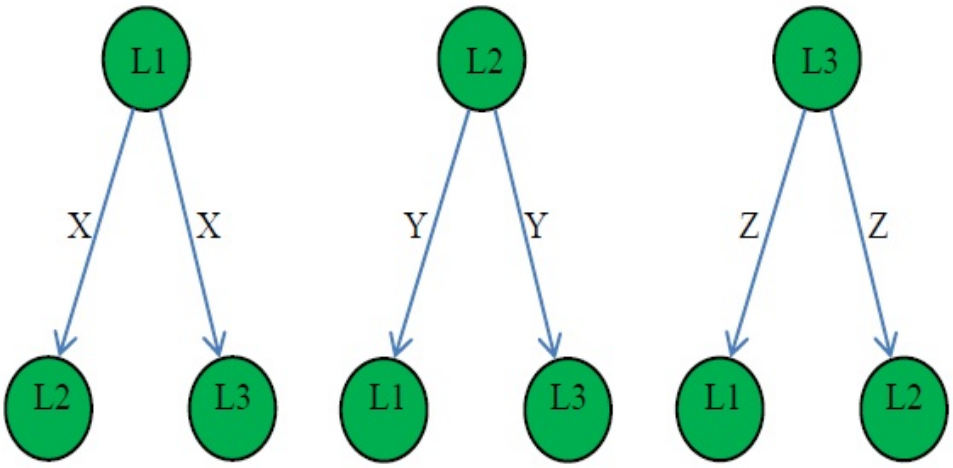

Figure 6: OM(0,t,v) state

5.3 Proof of Correctness:

The proof of correctness is done with the help of following two inductive arguments. We consider the cases where the commander is loyal or a traitor. If the commander is a traitor then it results in lot more chaos. We need to mainly prove that any two loyal lieutenants will end up with the same value after the algorithm stops irrespective of whether the commander is a traitor or loyal.

1. **Loyal Commander:** *For given m and k, and commander is loyal then Algorithm OM(m,t,v) works correctly if there are more than 2m + k generals and at most m traitors.*

   This can be prove by induction on k.

   - $OM(0, t, v)$ works if the commander is loyal (trivial case). So the our statement is true for $k = 0$.
   - Assuming it is true for $k - 1, k > 0$, we prove it for k.
   - In this algorithm, commander sends some value to $(n - 1)$ lieutenants who then forward it to remaining $(n - 2)$ lieutenants. After $m + 1$ rounds, each general applies *majority()* to the set of values they have received. The loyal general reports this value correctly while the traitorous general can lie. By hypothesis we have $n > 2m + k$. Among the $2m + k$ values taken in the calculation of majority at the end of the algorithm, the majority values come from the loyal lieutenants because $k > 0$.Hence taking the majority among the values received will give the correct solution.

2. **Commander may be loyal or traitor:** *For any m, Algorithm OM(m,t,v) satisfies Validity and Agreement conditions if there are more than 3m generals and at most m traitors.* [6]

   This can again be proved by induction on m.

   - $OM(0, t, v)$ satisfies Validity and Agreement (trivial case without any traitors). Assuming that the theorem is true for $OM(m-1, t, v)$ we prove it for $OM(m, t, v), m > 0$.

- Case 1: (Commander is loyal).
  We can directly apply the previous argument of "loyal commander" here. By taking k equal to m, we get OM(m) satisfying Agreement condition.
- Case 2: (Consider commander as a traitor).
  Here we have at most $m-1$ traitorous lieutenant. Since there are more than $3m$ generals, there are more than $3m-1$ lieutenants, and $3m-1 > 3(m-1)$. By applying induction hypothesis, $OM(m-1,t,v)$ satisfies Validity and Agreement conditions. Hence, any two loyal lieutenants get the same value for all other lieutenants in the end, due to which, any two loyal lieutenants get the same vector of values $< v_1.....v_{n-1} >$ and end up with same majority value in last step, proving Validity condition.

5.4 Message complexity for Oral Messaging Solution:

In the above examples we saw the way in which messages were exchanged between the generals. The tree structure in Figure 7 shows the number of messages being exchanged
In round 1, Commander sends a message to $(n-1)$ generals. In the next round these $n-1$

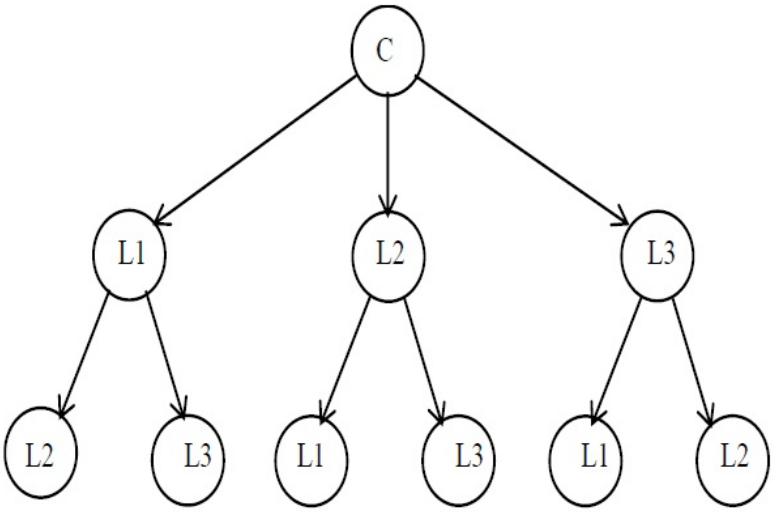

Figure 7: Message tree structure

lieutenants send the message to $n-2$ generals. Due to this the total messages sent is $(n-1)+(n-1)(n-2)$. Generalizing the above scenario, the recursive algorithm $OM(m,t,v)$ invokes $n-1$ separate executions of the algorithm $OM(m-1,t,v)$, each of which invokes $n-2$ executions of $OM(m-2)$ and so on. So the running time of the algorithm for oral messages can be calculated as follows:

- In $OM(m,t,v)$ the number of messages exchanged are $(n-1)$
- In $OM(m-1,t,v)$ the number of messages exchanged are $(n-1)*(n-2)$
- In $OM(m-2,t,v)$ the number of messages exchanged are $(n-1)*(n-2)*(n-3)\ldots$

Therefore for the total number of messages exchanged is given by $= O(n^m)$ Since m is at most $n/3$ we can say that the message complexity is $O(n^{n/3})$

5.5 Running time of Oral Messaging Solution:

We know there exists m traitors so we run the algorithm for m rounds where we assume that after m^{th} round the traitor generals cannot affect the result of the loyal generals. So in the $(m+1)^{th}$ round all the loyal generals go ahead with the agreed decision. Since this algorithm runs for $m+1$ rounds, the running time is $O(m)$.

6 Signed Message Algorithm:

This is a cryptographic protocol solution for Byzantine Generals' Problem. The messages exchanged here have the signatures of the general who has sent it. Looking at the previous algorithm we can see that a crucial part is the ability of traitorous Generals to simulate the actions of loyal Generals. But when the Generals sign the messages, it will restrict the traitors' ability to simulate the loyal generals action thus simplifying Byzantine Generals' Problem.

For the algorithm with signed messages we assume that tampering with the message is not allowed, a loyal general's signature cannot be forged and the authenticity of a general's signature can be verified by anyone. This is how the algorithm works: the commander sends a signed order to each of his lieutenants. Each lieutenant then adds his signature to that order and sends it to the other lieutenants, who add their signatures and send it to others, and so on. With the introduction of signed messages, we now have a solution for 3 generals' problem (see section 6.2 for the example). This algorithm copes with m traitors for any number of generals (the problem is however trivial if there are fewer than $m+2$ generals).

Let us first consider some notations: here $x : i$ denotes some value x signed by General i. If this is signed further by another General j then it looks like $x : i : j$. Each lieutenant maintains a set "`V`" containing properly signed orders he has received so far. We assume "`0`" to be the commander.

This algorithm assumes a function choice which is applied by the lieutenants to obtain one single order out of a set of orders. The requirements for this function are:

1. If V has only one value then it is trivial.
2. If the set V has more than a single element then the commander has to be a traitor. Hence the lieutenants follow the default order.
3. choice(ϕ) = RETREAT.

6.1 Algorithm:

We present the pseudo-code for the algorithm below.

Algorithm SM(m).

- Initially $V_i = \phi$.
- Commander(C) signs v and sends to every lieutenant(L) $(v : 0)$
- For each L_i :

  If received $(v : 0)$ from C and no other order, then

  – $V_i = v$.
  – Sign v and send to every other L $(v : 0 : i)$

  If received $(v : 0 : j_l : \ldots : j_k)$ and v is not in V_i, then

  – Add v to V_i.
  – if $k < m$, send $(v : 0 : j_l : \ldots : j_k : i)$ to all L other than $j_l ... j_k$.

(3) For each L_i : when no more messages, obey $choice(V_i)$.

The pseudo-code for the algorithm can be understood clearly with an example which is explained in the next subsection.

6.2 Working of Signed Messaging solution:

To understand the working of signed messages consider an example. As mentioned before that a solution to 3 generals' problem exists with signed messages, let us consider the case of 3 generals' problem with the commander being a traitor. Figure 8 shows the $SM(1)$ execution where the commander sends the $attack : 0$ order to lieutenant 1 and $retreat : 0$ to lieutenant 2 with his signature on it.

While executing $SM(0)$, as shown in Figure 9, both the lieutenants exchange their messages with their signatures on it. Lieutenant 1 sends $attack : 0 : 1$ and Lieutenant 2 sends $retreat : 0 : 2$.

After the messaging is completed we can see that the lieutenants eventually end up with $V_1 = V_2 = \{attack : 0, retreat : 0\}$. But unlike the previous case, the lieutenants will now spot the Commander being a traitor because he has his signature in front of two different orders. Hence they either can now take a decision together or fall back to the default value.

6.3 Proof of Correctness:

Here we need to again prove that two loyal lieutenants will end up with the same value when the algorithm stops. To do this, consider the following theorem.

Theorem: *For any m, Algorithm SM(m) solves the Byzantine Generals Problem if there are at most m traitors.*

Proof: We need to make sure that Validity and Agreement conditions are satisfied and any two loyal lieutenants will end up with the same value. Consider the two cases as shown below:

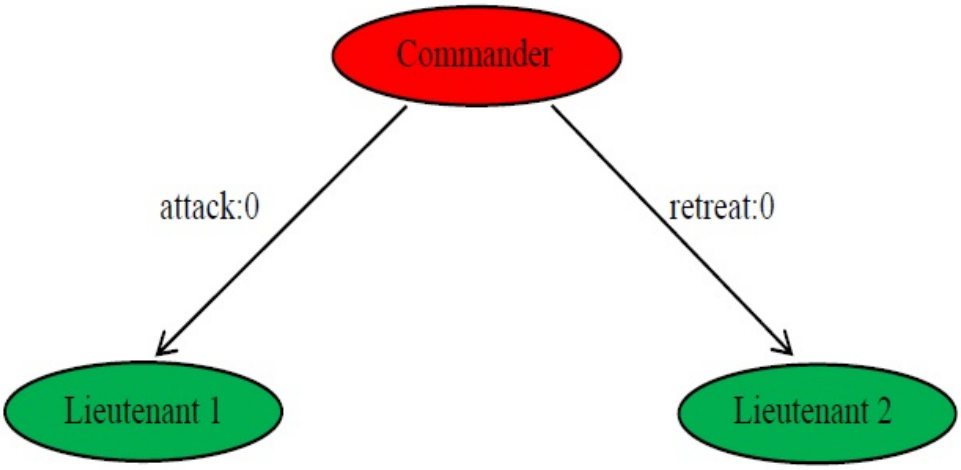

Figure 8: SM(1) for 3 generals' problem

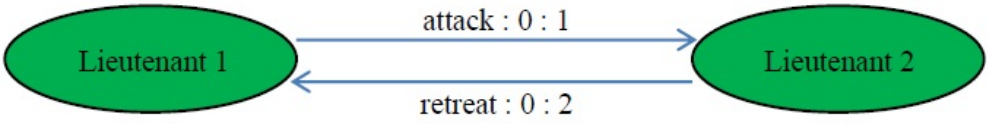

Figure 9: SM(0) for 3 generals' problem

1. Loyal Commander:
   - Initially commander sends $v : 0$ to $(n-1)$ lieutenants.
   - The loyal lieutenants will not receive any other order $v' : 0$ because the traitors cannot forge loyal commander's signature.
   - Hence all loyal lieutenants will have only one order which they will follow in the end.

   Validity follows from Agreement condition if the commander is loyal.

2. Traitorous Commander:
   - Two loyal lieutenants obey the same order if their sets have same values. Hence, to prove Validity condition we need to show that if an order v is added by a loyal lieutenant i in his set V_i then the same order v is added by another loyal lieutenant j in V_j.
   - If i receives an order $v : 0$, he sends it to j; so j receives it.
   - If i receives $v : 0 : j_1 : \ldots : j_k$, then, if j is one of the lieutenants who has already signed this order, then he must have already received it.
   - If not, consider 2 cases:

     If less than m lieutenants have signed the order, then i sends $v : 0 : j_1 : \ldots : j_k : i$ to j.

If m lieutenants have signed already, since at most $m-1$ traitorous lieutenants exists, one among the $j_1, \ldots, j_m$ is loyal and he must have sent v to j. Hence the loyal lieutenants will end up with same value and the algorithm is correct.

6.4 Message complexity of Signed Messaging Solution:

In the above examples we saw the way in which messages were exchanged between the generals. So to analyze the message complexity we consider the following reasoning. During this algorithm, every loyal lieutenant relays to every other lieutenant at most 2 orders. Therefore, the total number of messages exchanges is bounded by $2n(n-1)$, where n is the total number of generals. Thus we get the message complexity to be $O(n^2)$.

6.5 Running time of Signed Messaging Solution:

Since this algorithm runs for $m+1$ rounds, the running time is $O(m)$.

7 Asynchronous Byzantine Agreement:

7.1 Introduction:

The previous section dealt with algorithms which assumed synchronous system, were too slow to be used in practice. Also in the practical world it is difficult to realize a synchronous system. We would want the same fault tolerant systems for Internet which basically is an asynchronous system. The asynchronous model of distributed systems has no bounds on:

- Execution entity (the principles of distributed computing community uses the term "process")
- Execution latencies - i.e., arbitrarily long (but finite) times may occur between execution steps.
- Message transmission latencies - i.e., a message may be received an arbitrarily long time after it was sent
- Clock drift rates - i.e., process's local clocks may be arbitrarily unsynchronized. In other words, the asynchronous distributed system model makes no assumptions about the time intervals involved in any behaviors.

One such protocol which was developed in 1999 by Miguel Castro and Barbara Liskov is described in their paper "Practical Byzantine Fault Tolerance" [2].

7.2 Specifications:

The basic terminologies associated with this algorithm are discussed in brief:

1. State machine replication: The state machine approach is a general method for implementing a fault-tolerant service by replicating servers and coordinating client interactions with server replicas. The approach also provides a framework for understanding and designing the below discussed protocol.
2. Safety: All operations performed by faulty clients are observed in a consistent way by non-faulty clients and the algorithm does not rely on synchrony to provide safety.

3. Liveness: It is a property which guarantees that something good will eventually happen. If a network faults, we assume it is eventually repaired.

7.3 System Model:

The existing algorithm such as Rampart and SecureRing , were designed to be practical, but they relied upon the synchrony assumption for correctness, which is dangerous in the presence of malicious attacks. An attacker may compromise the safety of a service by delaying non-faulty nodes or the communication between them until they are deemed to be faulty and are discarded.

The assumptions of a system model to ensure that the algorithm works correctly are:

- Asynchronous distributed system where nodes are connected by a network. The network may fail to deliver messages, delay them, duplicate them, or deliver them out of order. All the elements are considered as state machine replicas. This means that each element duplicates the same initial state and runs the same version of the service code.
- Byzantine behavior of failed nodes.
- Assume cryptographic primitives. These cryptographic techniques prevent faulty processor from forging the signatures from other processors. These messages are signed by the sender and contain public key signatures, authentication codes, and message digests. All replicas are able to verify the signatures of the other replicas
- Very strong adversary that can coordinate faulty nodes, delay communication, or delay correct nodes in order to cause the most damage to the replicated service is assumed.
- Weak Synchrony is assumed which implies messages are retransmitted until they reach the destination. So the time between message deliveries is considered to be not indefinite.

7.4 Overview of Asynchronous Algorithm:

The algorithm makes use of state machine replication where each replica maintains service state and implements service operations. This algorithm mainly works in 5 rounds of communication (Request, Pre-prepare, Prepare, Commit and Reply). The communication happens between the client who requests for some information, primary replica which takes the requests (and acts like the Commander) and backup replicas (which are analogous to lieutenants). The number of replicas is assumed to be $3m + 1$ where m is the maximum number of faulty replicas. There are two requirements imposed on the replicas for this algorithm:

- The replicas must always produce same result when an operation is executed in a particular state and with given set of arguments. This means that they are deterministic.
- All the replicas must begin at the same time.

The entire process happens in something known as a *view* where some replica is considered as primary replica. The view changes occur when the primary replica fails. View will be helpful when the primary replica is faulty.

7.5 Details of Algorithm:

The algorithm mainly works as follows:

- Request Phase: The client sends a request to the primary replica in order to invoke a service operation.
- Pre-prepare Phase: The primary replica multi-casts the clients' request to all the backups.
- Prepare Phase: The backup replicas multi-cast the PREPARE message to all other replicas (including primary).
- Commit Phase: If the majority of the messages in previous phase match the initial message sent in Pre-prepare phase then the backups multi-cast a COMMIT message to all the other replicas. Once committed, the replicas now complete the operation and get a result.
- Reply phase: The replicas reply to the client with the result. The client waits to receive same result from $m + 1$ backup replicas.

Consider an example as in [2] where there are 3 backup replicas and 1 primary replica. Consider that the third replica is faulty. The messages exchanged during the 5 phases can be understood by Figure 10

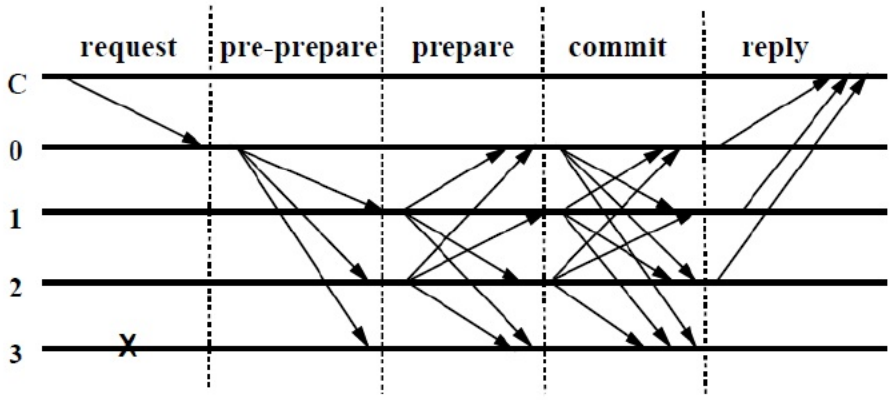

Figure 10: Example for asynchronous system with Byzantine failure

In this example, the 5 phases work as follows:

- Request phase: The client sends request to the primary replica (here we consider 0 as the primary replica).
- Pre-prepare phase: The primary replica forwards this request message to all the backup replicas.
- Prepare phase: The backup replicas multi-cast the PREPARE message to all other replicas (including primary). The faulty replica 3 does not participate as seen in Figure 10.
- Commit phase: All the non-faulty replicas (including primary), multi-cast the COMMIT message when the majority of the PREPARE messages in the previous phase match the initial request of the client. In this phase, again the faulty replica 3 does not participate.
- Reply phase: The non-faulty replicas reply back to the client. The client waits until it receives $m + 1$ same reply where m is the number of faulty replicas.

A detailed analysis of the algorithm can be found in the paper [2].

References

[1] Wikipedia: Byzantine fault tolerance. http://en.wikipedia.org/wiki/Byzantine_fault_tolerance/.

[2] M. Castro and B. Liskov. Practical byzantine fault tolerance. *Operating Systems Review*, 33:173–186, 1998.

[3] M. Castro and B. Liskov. Authenticated byzantine fault tolerance without public-key cryptography. Technical report, Technical Memo MIT/LCS/TM-589, MIT Laboratory for Computer Science, 1999.

[4] M. Castro and B. Liskov. A correctness proof for a practical byzantine-fault-tolerant replication algorithm. Technical report, Technical Memo MIT/LCS/TM-590, MIT Laboratory for Computer Science, 1999.

[5] B. Kapron, D. Kempe, V. King, J. Saia, and V. Sanwalani. Fast asynchronous byzantine agreement and leader election with full information. *ACM Transactions on Algorithms (TALG)*, 6(4):68, 2010.

[6] L. Lamport, R. Shostak, and M. Pease. The byzantine generals problem. *ACM Transactions on Programming Languages and Systems (TOPLAS)*, 4(3):382–401, 1982.

[7] M. Pease, R. Shostak, and L. Lamport. Reaching agreement in the presence of faults. *Journal of the ACM (JACM)*, 27(2):228–234, 1980.