CONSENSUS AND AGREEMENT

A consensus algorithm is a process that achieves agreement on a single data value among distributed processes or systems.

- Consensus algorithms necessarily assume that some processes and systems will be unavailable and that some communications will be lost.
- Hence these algorithms must be fault-tolerant.

Examples of consensus algorithm:

- Deciding whether to commit a distributed transaction to a database.
- Designating node as a leader for some distributed task.
- Synchronizing state machine replicas and ensuring consistency among them.

Assumptions in Consensus algorithms

- **Failure models:**

– Some of the processes may be faulty in distributed systems.

– A faulty process can behave in any manner allowed by the failure model assumed.

– Some of the well known failure models includes fail-stop, send omission and receive omission, and Byzantine failures.

– **Fail stop model:** a process may crash in the middle of a step, which could be the execution of a local operation or processing of a message for a send or receive event. it may send a message to only a subset of the destination set before crashing.

– **Byzantine failure model:** a process may behave arbitrarily.

– The choice of the failure model determines the feasibility and complexity of solving consensus.

- **Synchronous/asynchronous communication:**

– If a failure-prone process chooses to send a message to process but fails, then intended process cannot detect the non-arrival of the message.

– This is because scenario is indistinguishable from the scenario in which the message

– takes a very long time in transit. This is a major hurdle in asynchronous system.

– In a synchronous system, a unsent message scenario can be identified by the intended recipient, at the end of the round.

– The intended recipient can deal with the non-arrival of the expected message by assuming the arrival of a message containing some default data, and then proceeding with the next round of the algorithm.

- **Network connectivity:**

– The system has full logical connectivity, i.e., each process can communicate with any other by direct message passing.

- **Sender identification:**

– A process that receives a message always knows the identity of the sender process.

– When multiple messages are expected from the same sender in a single round, a scheduling algorithm is employed that sends these messages in sub-rounds, so that each message sent within the round can be uniquely identified.

- **Channel reliability:**

– The channels are reliable, and only the processes may fail.

- Authenticated vs. non-authenticated messages:

– With unauthenticated messages, when a faulty process relays a message to other processes

(i) it can forge the message and claim that it was received from anotherprocess,

(ii) it can also tamper with the contents of a received message before relayingit.

– When a process receives a message, it has no way to verify its authenticity. This is known as **un authenticated message or oral message or an unsignedmessage.**

– Using authentication via techniques such as digital signatures, it is easier to solve the

agreement problem because, if some process forges a message or tampers with the contents of a received message before relaying it, the recipient can detect the forgery or tampering.

- Thus, faulty processes can inflict less damage.

- **Agreement variable:**

- The agreement variable may be boolean or multivalued, and need not be aninteger.

- This simplifying assumption does not affect the results for other data types, but helps in the abstraction while presenting the algorithms.

Byzantine General problem

- The Byzantine Generals' Problem (BGP) is a classic problem faced by any distributed computer system network.
- Imagine that the grand Eastern Roman empire aka Byzantine empire has decided to capture a city.
- There is fierce resistance from within the city.
- The Byzantine army has completely encircled the city.
- The army has many divisions and each division has a general.
- The generals communicate between each as well as between all lieutenants within their division only through messengers.
- All the generals or commanders have to agree upon one of the two plans of action.
- Exact time to attack all at once or if faced by fierce resistance then the time toretreat all at once. The army cannot hold on forever.
- If the attack or retreat is without full strength then it means only one thing—Unacceptable brutal defeat.
- If all generals and/or messengers were trustworthy then it is a very simple solution.
- However, some of the messengers and even a few generals/commanders are

traitors. They are spies or even enemy soldiers.

- There is a very high chance that they will not follow orders or pass on the incorrect message. The level of trust in the army is very less.
- Consider just a case of 1 commander and 2 Lieutenants and just 2 types of messages- 'Attack' and 'Retreat'.

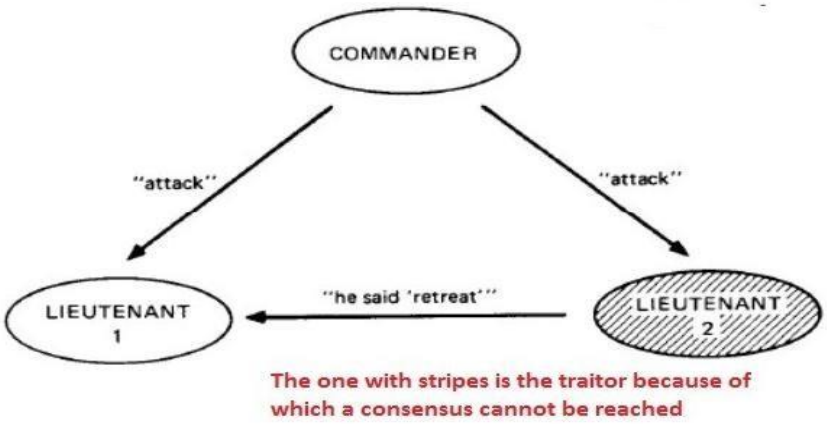

Fig : BGP algorithm

- In Fig, the Lieutenant 2 is a traitor who purposely changes the message that is to be passed to Lieutenant 1.
- Now Lieutenant 1 has received 2 messages and does not know which one to follow. Assuming Lieutenant 1 follows the Commander because of strict hierarchyin the army.
- Still, 1/3rd of the army is weaker by force as Lieutenant 2 is a traitor and this creates a lot of confusion.
- However what if the Commander is a traitor (as explained in Fig). Now 2/3rdof the total

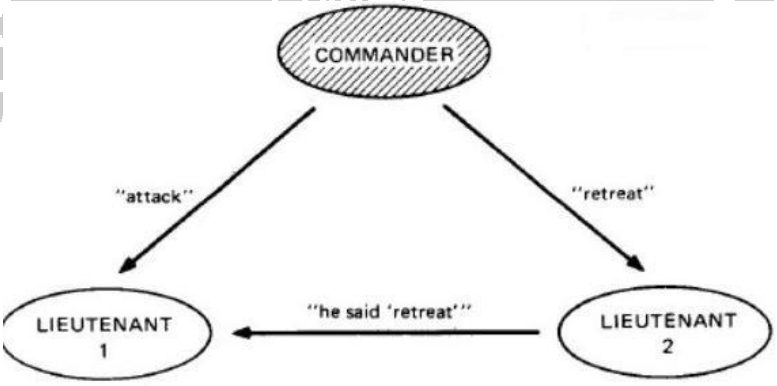

Fig: BGP algorithm

army has followed the incorrect order and failure is certain. After adding 1 more Lieutenant and 1 more type of message (Let's say the 3rd message is 'Not sure'), the complexity of finding a consensus between all the Lieutenants and the Commander is increased.

- Now imagine the exponential increase when there are hundreds of Lieutenants.

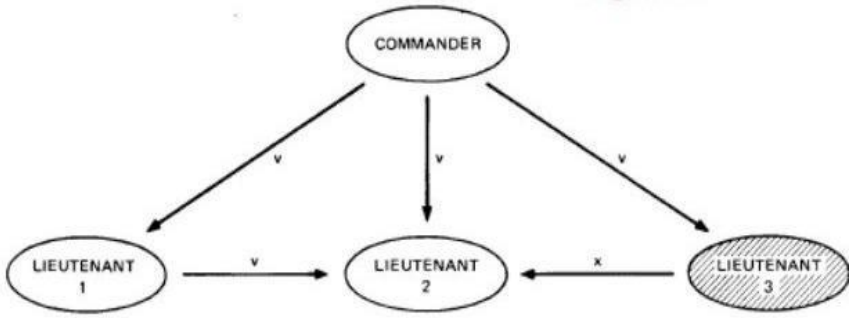

Fig : Adding one more lieutenant

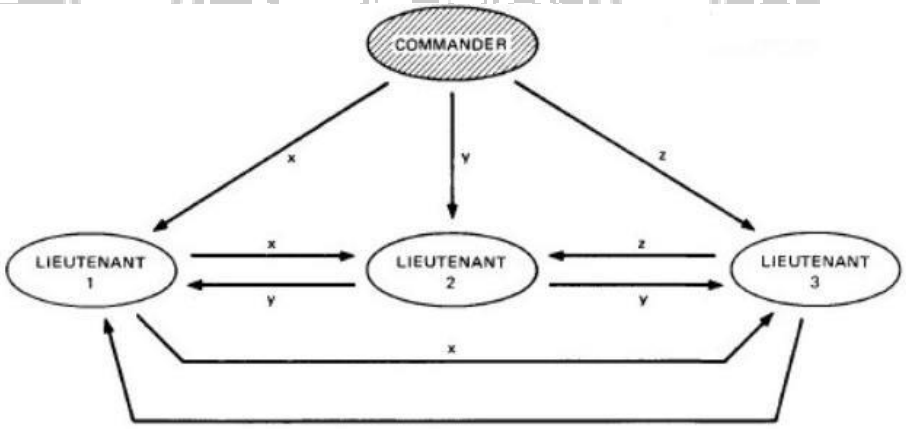

Fig : 1 commandant, 3 lieutenant and 3 types of messages

- This is BGP. It is applicable to every distributed network. All participants or nodes ('Lieutenant') are exactly of equal hierarchy. If agreement is reachable, then protocols to reach it need to be devised.

- All participating nodes have to agree upon every message that is transmitted between the nodes.
- If a group of nodes is corrupt or the message that they transmit is corrupt then still the network as a whole should not be affected by it and should resist this 'Attack'.

- The network in its entirety has to agree upon every message transmitted in the network. This agreement is called as **consensus.**

The Byzantine agreement problem requires a designated source process, with an initial value, to reach agreement with the other processes about its initial value, subject to:

- **Agreement:** All non-faulty processes must agree on the same value.
- **Validity:** If the source process is non-faulty, then the agreed upon value by all the non-faulty processes must be the same as the initial value of the

There are two other versions of the Byzantine agreement problem:

- Consensus problem
- Interactive consistency problem.
- A correct process is a process that does not exhibit a Byzantine behaviour.
- A process is Byzantine if, during its execution, one of the following faults occurs:

– **Crash:** The process stops executing statements of its program and halts.

– **Corruption**: The process changes arbitrarily the value of a local variable with respect to its program specification. This fault could be propagated toother processes by including incorrect values in the content of a message sent by the process.

– **Omission:** The process omits to execute a statement of its program. If a process omits to execute an assignment, this could lead to a corruption fault.

– **Duplication:** The process executes more than one time a statement of its program. If a process executes an assignment more than one time, this could lead to a corruption fault.

– **Misevaluation:** The process misevaluates an expression included in its program. This fault is different from a corruption fault: misevaluating an expression does not imply the update of the variables involved in the expression and, in some cases the result of an evaluation is not as-signed toa variable.

Consensus Problem

All the process has an initial value and all the correct processes must agree on single value. This is consensus problem.

Consensus is a fundamental paradigm for fault-tolerant asynchronous distributed systems. Each process proposes a value to the others. All correct processes have to agree (Termination) on the same value (Agreement) which must be one of the initially proposed values (Validity).

The requirements of the consensus problem are:

- **Agreement:** All non-faulty processes must agree on the same (single) value.
- **Validity:** If all the non-faulty processes have the same initial value, then the agreed upon value by all the non-faulty processes must be that same value.
- **Termination:** Each non-faulty process must eventually decide on a value.

Interactive Consistency Problem

All the process has an initial value, and all the correct processes must agree upon a set of values, with one value for each process. This is interactive consistency problem.

The formal specifications are:

- **Agreement:** All non-faulty processes must agree on the same array of values A [v1, …,vn].
- **Validity:** If process i is non-faulty and its initial value is vi, then all non faulty processes agree on vi as the ith element of the array A. If process jis faulty, then the non-faulty processes can agree on any value for A[j].
- **Termination:** Each non-faulty process must eventually decide on the array A.

The difference between the agreement problem and the consensus problem is that, in the agreement problem, a single process has the initial value, whereas in the consensus problem, all processes have an initial value.

RESULTS OF CONSENSUS PROBLEM

Some important facts to remember are:

- Consensus is not solvable in asynchronous systems even if one process can fail by crashing. Consensus is attainable for no failure case.
- In a synchronous system, common knowledge of the consensus value is also attainable.In asynchronous case, concurrent common knowledge of the consensus value is attainable.

The results are tabulated below. f indicates the number of processes that can failand n indicates the total number of processes.

S.No	Failure Mode	Synchronous System	Asynchronous System
1.	No failure	Agreement is attainable. Common knowledge is also attainable.	Agreement is attainable. Concurrent common knowledge is also attainable.
2.	Crash failure	Agreement is attainable. f < n process Ω(f+1) rounds	Agreement is not attainable.
3.	Byzantine (malicious) failure	Agreement is attainable. f<= floor((n-1)/3) Byzantine process Ω(f+1) rounds	Agreement is not attainable.

Solvable variants of agreement problem

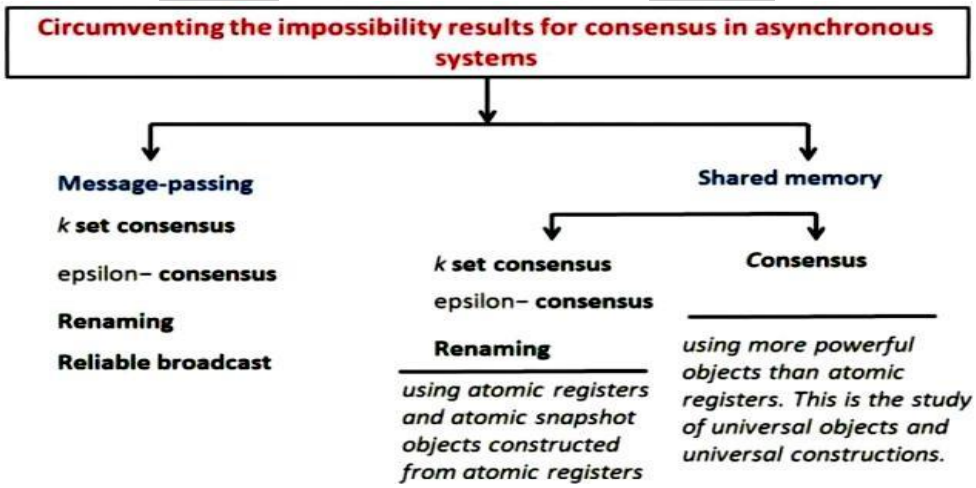

Fig : Circumventing the impossibility results

- A synchronous message passing system and a shared memory system can be used solve the consensus problem. The following are the weaker consensus problem in asynchronous system:
- **Terminating reliable broadcast:** A correct process will always get a message even if the sender crashes while sending. If the sender crashes while sending the message, the message may be even null, but still it has to be delivered to the correct process.
- **K-set consensus:** It is solvable as long as the number of crashes is less than the parameter k, which indicates the non-faulty processes that agree on different values, as long as the size of the set of values agreed upon is bounded by k.
- **Approximate agreement:** The consensus value is from multi valued domain. The agreed upon values by the non-faulty processes be within of each other.
- **Renaming problem:** It requires the processes to agree on necessarily distinct values.
- **Reliable broadcast**: A weaker version of reliable terminating broadcast (RTB), isthe one in which the terminating condition is dropped and is solvable under crash failures.

Solvable variants	Failure model and overhead	Definition
Reliable broadcast	Crash failures, $n > f$ (MP)	Validity, agreement, integrity conditions
k-set consensus	Crash failures, $f < k < n$ (MP and SM)	Size of the set of values agreed upon must be at most k
ϵ-agreement	Crash failures, $n \geq 5f + 1$ (MP)	Values agreed upon are within ϵ of each other
Renaming	Up to f fail-stop processes, $n \geq 2f + 1$ (MP) Crash failures, $f \leq n - 1$ (SM)	Select a unique name from a set of names

Fig : Solvable variants of agreement problem in asynchronous system