A Study on Efficient Consensus Algorithms for Byzantine-Prone Distributed Systems

A dissertation submitted

by

M.Nazreen Banu

for the award of the degree of

Doctor of Engineering

under the guidance

of

Prof. Koichi Wada

Department of Computer Science and Engineering

Nagoya Institute of Technology, Nagoya, Japan

March 2012

List of Publications

Journal Papers

1. Nazreen Banu, Taisuke Izumi and Koichi Wada, "Adaptive and Doubly-Expedited One-Step Consensus in Byzantine Asynchronous Systems", *Journal of Parallel Processing Letters*, 21(4): 461-477, 2011.

Conference Papers

1. Nazreen Banu, Taisuke Izumi and Koichi Wada, "Doubly-Expedited One-Step Byzantine Consensus", *Proceedings of the 2010 IEEE/IFIP International Conference on Dependable Systems and Networks(DSN), Dependable Computing and Communication Symposium(DCCS)*, pp. 373-382, Jul 2010.

2. Nazreen Banu, Samia Souissi, Taisuke Izumi and Koichi Wada, "An improved Byzantine Agreement Protocol for Synchronous Systems with Mobile Faults", *Proceedings of the 14th Korea-Japan Joint Workshop on Algorithms and Computation*, pp. 73-80, Jul 2011.

Technical Reports

1. Nazreen Banu, Taisuke Izumi and Koichi Wada, "Adaptive One-Step Byzantine Consensus", *Technical Report of IPSJ SIG*, 2010-AL-128, 2, pp. 1-9, Jan 2010.

Contents

List of Figures

List of Tables

Abstract

The *distributed consensus problem* is a fundamental and important problem in designing fault-tolerant distributed systems. In this problem, defined over a set of n processes, each process proposes a value, and all correct processes must decide on a common value, which is one of the proposed values. It has gained a leadership position as it is used as a key tool in many practical agreement problems such as atomic broadcast, view synchrony, etc. Several forms of the consensus problem have been proposed during the past three decades, considering various aspects like nature (crash or Byzantine) and computing power (bounded or unbounded) of the adversary, model of the underlying system (synchronous or asynchronous), etc. This dissertation studies the consensus problem in both asynchronous and synchronous systems with a more destructive adversary, namely computationally unbounded Byzantine adversary. Note that, any algorithm that solves the Byzantine consensus problem must work correctly no matter how the faulty processes behave, and hence assuming the Byzantine adversary leads to more complex and challenging algorithms.

In asynchronous systems, we carry out in-depth investigation on one-step Byzantine consensus algorithms that guarantee decision in one communication step (a communication step consists of a process sending a message to all other processes and receiving messages from them). In general, Byzantine consensus algorithms guarantee one-step decision only in favorable situations (e.g., when all processes propose the same value) and no one-step algorithm can support a two-step decision. We present a novel generic one-step Byzantine algorithm, called DEX, that circumvents these impossibilities using the adaptive condition-based approach. Algorithm DEX has two distinguished features: adaptiveness and double-expedition property. Adaptiveness makes it sensitive to only the actual number of failures so that it provides fast termination for a large number of inputs when there are fewer failures (a common case in practice). The other feature, double-expedition property facilitates the two-step decision in addition to the one-step decision. To the best of

our knowledge, double-expedition property is a new concept introduced by us, and DEX is the first algorithm having such a feature. In addition, we show that our algorithm is optimal in terms of the number of processes for one-step consensus.

In synchronous systems, we study the consensus problem with a powerful Byzantine adversary which can inject up to t malicious agents (such as viruses) that move from processes to processes at full speed and corrupt them in a dynamic fashion. This variant of the consensus problem is popularly known as the mobile Byzantine consensus problem. In a previous result, Garay has shown that $n > 6t$ (n is the total number of processes, and t is the number of mobile faults) is sufficient to solve the problem even in the presence of strong agents that can move with full speed (in the sense that each agent can take a movement in every communication step) and make corrupted processes forget that they run the algorithm by erasing their local memories (as a result, after recovery a process must learn the current state of computation including the code from other processes). Many following results have improved the above result, but with some additional assumptions. The question, whether this result can be improved without any additional assumption, remains open. In this dissertation, we first answer this question, by providing an algorithm *MBC* that works with $n > 4t$ under the same settings. In addition, we study the mobile Byzantine consensus a new system model in which agents do move at full speed, but they do not erase the local memories of processes, and the processes are allowed to restart their algorithms immediately after their recovery with some predefined state. This new model mainly differs from Garay's model in the sense that it allows recovering processes to rejoin the on-going execution without learning the current state of computation, while in Garay's model processes must learn the current state of computation before reintegration. We show that our algorithm *MBC* also works in this new model with minor modifications.

Chapter 1

Introduction

1.1 Background

"Can you live without ATM and Internet? What do you do, if your ATM machine shows incorrect balance? or your email access is completely denied?"

World Wide Web and Automatic banking(Teller Machine) systems are the best examples for distributed systems that play vital roles in our day-to-day lives. A distributed system is a collection of autonomous (probably heterogeneous) computers connected through a network and distribution middleware, in order to provide certain kind of services. The distributed systems provide several good properties such as high performance, scalability, openness, etc., which enable them to overcome many physical limitations of centralized systems. However, if not designed carefully, a distributed system can decrease the reliability of computation and/or functionality. In fact, designing a properly functioning distributed system is notoriously difficult. One of the key factors causing the difficulty is *faults*, which means that computing entities can fail independently, leaving some components operational while others are not. Leslie Lamport once amusingly defined distributed systems as such:

A distributed system is one in which the failure of a computer you didn't even know existed can render your own computer unusable.

In large scale systems, it is unavoidable that the system is subject to faults.

Hence, *fault tolerance*, which is the ability of a system to perform its functions correctly even in the presence of partial failures, is highly required.

The *distributed consensus problem* (in what follows, we simply call the consensus problem for short) is a fundamental and important problem that has to be solved in designing fault-tolerant distributed systems. This problem can be stated informally as: how to ensure that a set of distributed processes achieve agreement on a value or an action despite a number of faulty processes. This problem is interesting both in theoretical and practical aspects. From theoretical point of view, the significance of the consensus problem derives from several other distributed systems problems being reducible or equivalent to it. Examples are atomic broadcast [21, 24, 47], non-blocking atomic commit [46], group membership [46] and state machine replication[65]. The relations between the consensus problem and other distributed ones are important because the consensus problem is deeply studied problem and many results stated for the consensus problem automatically apply to these other ones. From a system perspective, replication of components and services is an important paradigm that can be employed for information protection in critical applications, and consensus plays a fundamental role in many replication algorithms. Some example solutions based on these ideas are: Bessani et al. [9] and Yin et al. [73] use replication to implement fault- and intrusion-tolerant firewall devices; Cachin et al. [16] and Castro and Liskvo [18], Chun et al. [22] and Veronese et al. [71] propose replication algorithms to implement highly resilient services, like data historians or DNS (essential for the Internet).

Algorithms that solve consensus vary much depending on the assumptions that are made about the system. This dissertation considers the systems that experience *Byzantine faults*, more destructive faults which do not put any constraints on how processes fail. Algorithms based on this system model are expected to work correctly no matter how faulty processes behave. Byzantine consensus algorithms are highly essential for practical systems to deal with malicious attacks or situations in which faults are difficult to characterize. For instance, an attacker might modify the behavior of a process it controls in order to change the outcome of the consensus algorithm. However, assuming Byzantine faults leads to more complex and challenging algorithms.

In this dissertation, we study the consensus problem in Byzantine-prone distributed systems and contribute significantly for the advancement of the state-of-the-art research on this problem. This chapter is now molded in the following manner: First, we give an overview of our problem which starts by tracing the genesis of the problem. Then, we list different models in which the problem has been studied so far and can be looked at in future. Next, we present various metrics that are used for evaluating Byzantine consensus algorithms. After that, we present the models and metrics of our interest. Next, we provide the related work and our objectives. We then emphasize on our contributions in this thesis and their impacts on the literature. Lastly, we describe the chapter wise organization of this thesis.

1.1.1 Byzantine Consensus Problem: An Overview

The problem of Byzantine consensus (popularly known as Byzantine General's problem) is introduced by Lamport et al. in [52]. The most basic and commonly used form of Byzantine consensus is as follows: Byzantine consensus, among a set of n processes each having a private input value, allows all non-faulty processes to reach an agreement on a common value even if faulty processes try to prevent agreement among the nonfaulty processes. The faulty behavior may range from simple mistakes to total breakdown to skillful adversarial talent. For instance, a faulty process can send messages when it is not supposed to, make conflicting claims to other processes, act dead for a while and then revive itself, or the faulty processes can act maliciously against the protocol in an coordinated fashion. Attaining agreement on a common value is difficult as one does not know whom to trust.

This problem can be informally stated in terms of three properties: each process proposes a value, and the non-faulty processes have to decide (*Termination property*) on a common output value (*Agreement property*) that has to be related to the set of input values (*Validity property*). This problem has drawn much attention over the years and many aspects of the problem have been studied considering various models [1, 2, 4, 7, 8, 10, 11, 12, 13, 17, 26, 31, 32, 33, 37, 40, 51, 53, 57, 69, 70].

1.1.2 Various Models for Studying Byzantine Consensus

The Byzantine consensus problem may take many different forms, depending on the computational model (that talks about the attributes of the processes and the communication channels) and the adversary model (that captures the nature, capacity and computing power of the adversary).

Computational model

The prominent attributes of the processes and the communication channels that lead to the various classifications of Byzantine consensus are discussed below.

- **Synchrony of Network (Synchronous or Asynchronous or Hybrid):** Synchrony divides the systems into three types: synchronous, asynchronous and hybrid. In synchronous systems [53], all processes have access to a common global clock and there exists a known upper bound on the relative speed of processes and the message transmission delay. Processes execute in lockstep: the execution is divided into rounds and in each round, each process sends a message to all other processes, receives the messages sent to it and does some local computation.

  In asynchronous systems, there is no global clock and there is no fixed upper bound on the relative processes speed and the message latency. In particular, the messages may be received in an order different from the one in which they were sent. Thus, in asynchronous systems [53], the inherent difficulty in designing a protocol comes from the fact that when a process does not receive an expected message then it cannot decide whether the sender is corrupted (and did not send the message at all) or the message is just delayed in the network. So a process can not wait to consider the values sent by all processes before commencing its computation at any particular step, as waiting for all of them can turn out to be endless. Due to this, the protocols in asynchronous systems are generally involved in nature and require new sets of primitives.

There is another class of system called hybrid system that exercises the properties of synchronous and asynchronous systems in many different ways. There are at least three different notions for hybrid system available in the literature: (a) a hybrid system that allows a few synchronous rounds followed by a fully asynchronous communication; (b) a hybrid system [18], that behaves asynchronously for some time interval, but that eventually stabilizes and starts to behave more synchronously, and (c) a hybrid system that alternates between synchrony and asynchrony [10]. The synchrony reflects some effects on the behavior of adversary as well. In asynchronous systems, the adversary is given the power to schedule the delivery of all messages in the network.

- **Medium of Communication (Uni-cast channel or Multi-cast channel or Broadcast channel):** In any protocol, the processes communicate with each other over channels where channels can be uni-cast/point-to-point (one-to-one), multi-cast (one-to-many) and broadcast (one-to-all). The uni-cast/point-to-point channel enables both way communication between two processes. Multi-cast channels allow a process to send some message identically to a subset of processes in the network. The broadcast channel allows a process to send some message identically to all other processes in the network. Uni-cast and broadcast channels are two extreme cases of multi-cast channels. We may consider the channels to be undirected and directed. Most of the literature on Byzantine consensus assume the existence of pairwise point-to-point undirected channels among the processes and often, broadcast channels are also assumed. In many other cases, in the absence of broadcast channel, it is simulated by executing broadcast (a variant of Byzantine consensus) protocol. There is little work in the literature that considers multi-cast channels [23, 61]. In general, when we say channel (or link), we will usually mean uni-cast or point-to-point channel.

- **Network Topology (Complete or Incomplete):** The topology of the network can be complete or incomplete. In a complete network, every pair of processes are directly connected, while in an incomplete network, the connectivity can be limited. Except a very few attempts

[1, 28, 29, 52], most of the work on Byzantine consensus consider complete networks.

- **Control over Channels (Reliable or Unreliable):** We distinguish two levels of control over the channels: reliable and unreliable. In reliable channels, messages are neither lost, corrupted nor duplicated. In unreliable channels, messages may be either dropped or corrupted, or spurious ones may be generated.

Adversary Model

Various models of Byzantine consensus can be obtained based on the kinds of adversary. Some of the features which characterize the adversary are discussed below.

- **Computational Resources (Bounded or Unbounded):** The computational resources at the disposal of the adversary may be limited to probabilistic polynomial time as in cryptographic settings [34]. On the other hand adversary may have unbounded computing power as in information theoretic settings [5].

- **Mobility (Static or Adaptive/Dynamic or Mobile/Proactive):** Depending on the point in time when the adversary is allowed to corrupt processes, adversary can be of three types: static, adaptive/dynamic and mobile/proactive. If the adversary decides on the set of processes that it would corrupt before the protocol begins its execution, then that adversary is referred to as static adversary [6]. Thus, the set of corrupted processes is fixed (but typically unknown) during the whole computation. More generally, the adversary may be allowed to corrupt processes during the protocol execution, depending on the information gathered so far. Such an adversary is called adaptive or dynamic. Thus, an adaptive or dynamic adversary [38] chooses which processes to corrupt as the computation proceeds. In both the above cases, once a processes is corrupted, it remains corrupted for the rest of the protocol execution. Like an adaptive adversary, a mobile adversary can corrupt processes at any time, but it can also release corrupted processes and

regain the capability to corrupt further processes. Thus, an adversary is mobile [15, 43] if it can corrupt, in an adaptive way, a different set of processes at different times during the execution. That is, a process once corrupted need not remain so throughout. Mobile adversaries model, for example, the virus attacks.

- **Corruption Capacity (Threshold or Non-threshold):** The number of processes that an adversary can keep corrupted at any given instance of time is its corruption capacity. There are two different ways for specifying the number of corrupted processes, viz. threshold and non-threshold. In the threshold specialization [52], the number of corrupted processes, at any given time, is limited to at most t (a threshold). The non-threshold specialization is a generalization of the threshold one. In the non-threshold specialization [41], an adversary structure, which is a set of subsets of the processes, is used where the adversary is permitted to corrupt the processes of any one arbitrarily chosen subset in the adversary structure.

- **Scheduling Capacity (Message-Oblivious or Message-Conscious):** In asynchronous systems, the adversary may be able to schedule the messages and processes. The message-oblivious adversary [42] is a weak adversary which cannot look at the internal state of processes or the contents of messages. Hence, it neither controls the delivery order of messages nor schedules correct processes. The message-conscious [66] adversary has full knowledge of the internal states of faulty processes and the contents of messages in the network. It can dynamically decide which process takes the next step by arbitrarily delaying messages between processes. However, this adversary operates under the following restrictions: the final schedule must be fair, meaning that all correct processes take infinitely many steps, and the messages are eventually delivered.

1.1.3 Efficiency Metrics for Byzantine Consensus Algorithms

The efficiency of Byzantine consensus algorithms is usually evaluated by using four different metrics: resilience, time complexity, communication complexity and computation complexity. Remember that, the total number of processes participating in an algorithm is often designated with the letter n.

1. **Resilience:** It is the maximum number of corrupted processes that the algorithm can tolerate and still satisfy its properties.

2. **Time Complexity:** The time complexity is the maximum number of rounds taken for the execution of the algorithm. A round consists of a process sending a message, receiving one or more messages sent by other processes and performing some local computation. An algorithm is called round efficient if its round complexity is (at most) polynomial in n.

3. **Communication Complexity:** It is the total number of bits communicated by the correct processes in the protocol. A protocol is called communication efficient if its communication complexity is (at most) polynomial in n.

4. **Computation Complexity:** It is the computational resources required by the correct processes during a protocol execution. An algorithm is called computationally efficient if the computational resources required by each correct process are (at most) polynomial in n.

1.2 The Models and Metrics of Our Interest

In this dissertation, we are interested in investigating Byzantine consensus in two system models (that mainly differ in synchrony assumptions and mobility of faults) with the aim to design protocols that minimize the two important parameters, namely, the total number of processes and the required number of communication rounds. Table 1.2 describes the attributes of our systems. In brief, we first concentrate on designing fast-terminating highly-resilient

Byzantine protocols for asynchronous systems prone to adaptive computationally unbounded Byzantine adversary. Next, we focus on designing mobile Byzantine consensus protocols for synchronous systems that provide better resilience than the existing protocols. We will discuss about our models more elaborately and will present the formal definitions of the consensus problem in respective models in individual chapters of this thesis.

	System Model1	**System Model2**
Computational Model:		
1. Synchrony of Network	Asynchronous	Synchronous
2. Medium of Communication	Point-to-point channels	Point-to-point channels
3. Network Topology	Complete network	Complete network
4. Control over Channels	Reliable Channels	Reliable Channels
Adversary Model:		
1. Mobility	Adaptive	Mobile
2. Computational Resources	Unbounded computing power	Unbounded computing power
3. Corruption Capacity	Threshold	Threshold
4. Scheduling Capacity	Message-conscious	——

Table 1.1: Attributes of our systems.

1.3 Related Work and Our Objectives

We divide the literature survey into two parts: the first part focuses on the works in asynchronous systems that motivated us to investigate *one-step Byzantine consensus algorithms* (which guarantee consensus in one round). The second part presents the works that stimulated us to explore *mobile Byzantine consensus* in synchronous systems.

1.3.1 One-Step Byzantine Consensus

It is far from a trivial task to design Byzantine consensus algorithms for asynchronous systems since these algorithms have to deal with the uncertainty and unpredictability created by the combined effect of Byzantine behavior and asynchrony. A more discouraging impossibility result (called FLP result [39]) shows that the consensus problem can not be solved deterministically

in asynchronous systems even if a single process can crash (when a process crashes, it stops prematurely and makes no operation subsequently, and hence crash faults are less severe than Byzantine faults). This result has led to a large number of works that attempt to circumvent it by slightly modifying the system model. Examples include: randomization [4, 60], failure detectors [21, 54], partial synchrony [30, 36], hybridization/wormholes [25, 58], condition-based approach [55].

The number of communication steps (usually called rounds) required to achieve consensus is one of the important measures of consensus algorithms. It has been proved that any consensus algorithm requires at least two communication steps for decision even in failure-free executions [50]. This lower bound often becomes a dominant part of the performance overhead imposed on consensus-based applications. However, this fact does not imply that the two-step lower bound is incurred for every input (an input to consensus algorithms is defined as a n-tuple consisting of all proposed values). For example, it does not hold for the case where all processes propose the same value. Furthermore, in typical runs of consensus-based applications, the consensus algorithm often receives such good inputs. For instance, consider a replicated state machine: the replicated servers need to agree on the processing order of the update requests; if a client broadcasts its request to all servers and there is no contention, then all servers propose the same request as the candidate they will handle next; practically, it is not so often that two or more concurrent update-requests arise for the same data object. This observation induces an interest in one-step decision for good inputs.

The first attempt to circumvent this two-step lower bound is done by Brasileiro et al. [14]. They propose a general framework to convert any crash-tolerant algorithm into the one that solves the consensus for any input and especially terminates in one step when all processes propose the same value. In other results [27, 35], the notion of one-step decision is considered in combination with the other schemes such as randomization and failure detectors.

An interesting aspect of one-step decision schemes is to characterize the situations where one-step decision is possible. The first investigation in that aspect is considered by Mostefaoui et al. [56] in synchronous systems prone

to crash failures. It applies the *condition-based approach* for obtaining a good one-step decision scheme. This result is extended by Izumi and Masuzawa [49]. They give the complete characterization of conditions that make one-step decision possible.

While all the above results are considered on crash failure model, Friedman et al. [42] have presented an asynchronous Byzantine consensus protocol that guarantees one-step decision when all processes propose the same value. But, their protocol can not tolerate a message-conscious adversary, which can arbitrarily re-order messages. In the following work *BOSCO* [66], the authors have devised a simple Byzantine one-step algorithm that can defeat a strong message-conscious adversary. This work has shown two variants of one-step Byzantine consensus problem, weak and strong ones, and a lower bound on the number of processes needed for each. The weak one-step guarantees one-step decision only when all the processes propose the same value and no process is faulty, but the strong one-step must guarantee it in any situation in the case of a common proposed value, regardless of the number of faulty processes. In addition, the authors proved that the assumptions $n > 5t$ and $n > 7t$ are necessary for weak and strong one-step Byzantine consensus, respectively, where n is the number of processes and t is the maximum number of faulty processes.

The two major drawbacks of one-step Byzantine algorithms are the lack of adaptiveness to the actual number of failures and the impossibility of zero-degradation [27, 45]. In short, adaptiveness is an interesting property of consensus algorithms, which makes the algorithms sensitive only to the number of failures k ($k \leq t$) that actually occur in a given run, rather than on the (total) number of failures t that can be tolerated. Adaptive algorithms are very appealing in practice since they guarantee some additional good properties when there are fewer failures. In practice, failures rarely occur. Hence, in the context of one-step algorithms, it is reasonable to expect that, in runs where fewer failures occur, an algorithm should provide one-step decision for a large number of inputs than in runs with many failures. Unfortunately, all previous one-step Byzantine algorithms are non-adaptive: their efficiencies depend only on the total number of failures that can be tolerated, rather than the actual number of failures. Consequently, these algorithms behave

pessimistically in runs where there are fewer failures and provide one-step decision for very few inputs. Similarly, the zero-degradation is another important feature of consensus algorithms based on failure detectors, which always guarantees the best complexity (i.e., two step decision) in *stable* runs where the failure detector does not make any mistakes and its output is stable. Existing results [27, 45] have proved that no consensus algorithm can be simultaneously one-step and zero-degrading. It means that, one-step algorithms are optimal (i.e., one-step) for some particular case (e.g., when all processes propose the same value), but need at least three communication steps starting from other configurations. It has also been shown that achieving both one-step decision and zero-degradation, even with crash failures, requires a stronger assumption about failure detections such as the existence of eventually perfect failure detector $\diamond P$ [27]. These results raise a natural question:

Can we design a Byzantine algorithm that simultaneously be adaptive, one-step and zero-degrading, without having any additional assumptions?

The first objective of our dissertation is to present an adaptive one-step Byzantine consensus algorithm that provides fast termination for a large number of inputs when there are fewer failures, and ensures the two-step decision in addition to the one-step decision.

1.3.2 Mobile Byzantine Consensus

Computer viruses pose one of the central problems in distributed systems. In many cases, reaching a common agreement among fault-free processes in the presence of moving malicious agents (viruses), is highly required. This subsection considers the problem of reaching and maintaining agreement among a set of correct processes in the presence of a powerful adversary that distributes malicious agents which can corrupt processes at any time and move from one process to another. This problem is referred to as the mobile Byzantine consensus problem.

In a previous work [43], Garay has studied the power of disruption of mobile Byzantine faults as a function of the speed with which they can traverse

the network (called roaming pace). The roaming pace ρ denotes the minimal amount of time (i.e., the number of rounds) that has to elapse between the time at which an agent leaves a process, and the time at which it starts to corrupt another process. For example, $\rho = 3$ means that an agent takes at least 3 rounds to hop from one process to another. Also, Garay [43] proposed two Byzantine consensus protocols for synchronous systems: The first one assumes $n > 6t$ and can cope with malicious agents that move at full speed (i.e., $\rho = 1$); The second one assumes $n > 4t$ and deal with agents that move at half speed (i.e., $\rho = 2$). These protocols run in $O(n)$ communication rounds. In his system model, an agent can move from one process to another at any time during a round and can make corrupted processes forget that they run the algorithm by erasing their local memories (as a result, after recovery a process must learn the current state of computation including the code from other processes). In addition, the agent is allowed to corrupt a new process before the currently corrupted process recovers.

In the following work, Burhman et al. [15] proposed an optimal Byzantine consensus algorithm with $n > 3t$ for full speed agents. However, their model has two additional assumptions: agents can move from one process to another only through messages (i.e., the migration of the agents is possible only during the send operations), and a faulty process must recover and learn the current state of computation before another process can fail instead of it. In a recent work, Biely et al. [10] proposed a mobile Byzantine consensus algorithm for partially synchronous system for $n > 3t$, with the same assumption that a faulty process must recover and learn the current state of computation before another process can fail instead of it. However, this algorithm terminates only when all processes have recovered. In a different work, Schmid et al. [64] proposed impossibility results and lower bounds for mobile Byzantine consensus, but for link failure model.

The question that remains open is the following:

Can we improve the result of Garay [43]($n > 6t$) without adding any additional assumptions?

The second objective of our dissertation is to investigate the mobile Byzantine consensus problem in Garay's model to find the answer for the

above question. In addition, we aim at studying this problem in a new model in which malicious agents can move at full speed, but they can not erase the local memories of processes, and a corrupted process can restart the algorithm immediately with some predefined state after its recovery. Note that, this new model differs from Garay's model in the sense that it allows the recovering processes to rejoin the on-going execution without knowing the current state of computation, while in Garay's model processes must learn the current state of computation before rejoining.

1.4 Our Contribution: An Overview

This section is devoted for the description of our contributions. In this thesis, we first study the Byzantine consensus in asynchronous systems with better one-step decision schemes. Then, we focus on mobile Byzantine consensus in synchronous systems with the aim to present an algorithm that provides better resiliency than the existing algorithms.

1.4.1 Adaptive and Doubly-Expedited One-Step Byzantine Consensus

As seen in Section 1.3.1, the two major drawbacks of the existing one-step Byzantine algorithms are the lack of adaptiveness to the actual number of failures (due to which, they provide one-step decision for very few inputs) and the impossibility of zero-degradation (which says that no one-step algorithm can support two-step decision). In this thesis, we design an efficient one-step Byzantine algorithm that circumvents these drawbacks using *adaptive condition-based approach*[48]. This approach has taken its root from *condition-based approach* [55], which says that if a problem is not solvable for general inputs, it is better to solve it for a restricted set of inputs. In condition-based approach, a condition represents some restriction to inputs. In the context of consensus problem, it is defined as a subset of all possible input vectors (whose entries correspond to the proposal of each process). For instance, $C_t^{prv(m)}$ is a privileged-value-based condition [14] defined for t crash failures which includes all input vectors where the privileged value m

appears more than t times (the privileged value is one of the proposal values which is given high priority among the set of all possible proposal values). A condition-based algorithm instantiated with a condition guarantees some good properties if the actual input vector belonging to that condition. Mostefaoui et al. [55] have presented a class of conditions, called *d-legal conditions*, and a condition-based algorithm which solves consensus in asynchronous systems where at most d processes can crash. In adaptive condition-based approach, the set of input vectors belonging to a condition dynamically changes according to the actual number of faulty processes (typically, a small number of faults allow the condition with a large number of inputs). The adaptiveness property of a condition (defined for t failures) can be characterized by a *condition sequence*, which is defined as a sequence of $t+1$ conditions $(C_0, C_1, ..., C_k, ..., C_t)$ such that $C_k \supseteq C_{k+1}$ for any $k(0 \leq k \leq t-1)$ and the k-th condition is a subset of all possible input vectors which guarantee some good properties when the actual number of faulty processes is at most k. From the definition, the condition C_{k+1} is a subset of C_k. In the context of one-step consensus, it means that if an algorithm is instantiated with a one-step condition sequence, in runs where at most k processes are faulty, it provides one-step decision for more number of inputs than in runs where at most $k+1$ are faulty. In a previous result [49], Izumi et al. have presented such a one-step condition sequence and an adaptive condition-based one-step algorithm for crash-failure model [49], however, there is no result to apply it in Byzantine-failure model. In this dissertation, we construct one-step condition sequences for Byzantine failure models such that each sequence $S^1 = (C_0^1, C_1^1, ..., C_k^1, ..., C_t^1)$ where the k-th condition is the set of input vectors that support one-step decision when at most k processes are Byzantine. A condition-based algorithm instantiated with a condition-sequence S^1 can terminate in one-step when the actual input vector belonging to the k-th condition and at most k processes exhibit Byzantine behavior.

Another interesting point is that, the zero-degradation impossibility result does not say that the two-step decision is impossible for any input. It means that there are good inputs for which the one-step consensus algorithms can provide two-step termination (Remember that, previous one-step algorithms provide one-step termination when the actual input vector belonging to the

given condition, otherwise they require at least three communication steps for termination. That is, they never provide two-step decision). In our work, we identify the (good) inputs that support two-step decision and construct two-step condition sequences such that each sequence $S^2 = (C_0^2, C_1^2, ..., C_k^2, ..., C_t^2)$ where the k-th condition is the set of all input vectors that allow two-step decision when at most k processes are Byzantine. This helps us to realize a doubly-expedited consensus algorithm, which equips a two-step condition sequence in addition to a one-step condition sequence. Since our algorithm is instantiated with a pair of condition sequences (S^1, S^2), each for one-step and two-step decisions respectively, in any execution where at most k processes are Byzantine, it provides one-step and two-step decisions if the actual input vector belonging to C_k^1 and C_k^2 respectively. In addition, in the same way as [49], we can equip our doubly-expedited algorithm with an underlying consensus primitive (which can be any algorithm that solves Byzantine consensus in asynchronous systems, but takes more than two steps). As a result, our algorithm can terminate for any input that even lies out of the (given one-step and two-step) conditions, but not in one or two steps.

To explain the concept of double-expedition more clearly, here we present an example. Using the privilege-value based condition $C_t^{prv(m)}$, let us construct a condition-sequence pair for doubly-expedited Byzantine consensus as follows:

$$< S_1, S_2 >= ((C_0^{prv(m)^1}, C_1^{prv(m)^1}, ..., C_k^{prv(m)^1}, ..., C_t^{prv(m)^1}),$$
$$(C_0^{prv(m)^2}, C_1^{prv(m)^2}, ..., C_k^{prv(m)^2}, ..., C_t^{prv(m)^2}))$$

where $C_k^{prv(m)^1}$ and $C_k^{prv(m)^2}$ contain input vectors where the privileged value m appears more than $3t + k$ and $2t + k$ times respectively. Let $\mathcal{A}$ be an algorithm instantiated with this condition sequence pair for values $t = 1$, the set of all possible proposal values $\mathcal{V} = \{0, 1, 2, 3\}$ and $m = 3$. Consider the four input vectors $I_1 =< 0, 1, 2, 2, 3 >$, $I_2 =< 0, 1, 3, 3, 3 >$, $I_3 =< 1, 3, 3, 3, 3 >$, $I_4 =< 3, 3, 3, 3, 3 >$. When the actual number of failures is 1, $\mathcal{A}$ provides one-step decision for the vector I_4 since I_4 is contained in $C_1^{prv(m)^1}$, but it terminates in two-steps for vector I_3 since I_3 is in $C_1^{prv(m)^2}$. When there are no failures, $\mathcal{A}$ guarantees one-step decision for vectors I_3 and I_4 since I_3 and I_4 are in $C_0^{prv(m)^1}$, and provides two-step termination for I_2 since I_2 is contained in $C_0^{prv(m)^2}$. For the vector I_1, the algorithm terminates, but not in

one or two steps.

In brief, this dissertation contributes an adaptive doubly-expedited Byzantine consensus algorithm *DEX* whose distinguished features can be summarized as follows:

- In our construction, we show a generic framework of the algorithm based on the notion of adaptive condition-based approach. To attain the double-expedition property in adaptive manner, the framework is instantiated with a pair of condition sequences, each of which corresponds to the situations of one-step and two-step decisions respectively. We also show sufficient criteria, say *legality*, for the condition-sequence pair such that the doubly-expedited algorithms can be instantiated by using any condition-sequence pair satisfying them.

- We propose two examples of legal condition-sequence pair, called *frequency-based pair* and *privileged-value-based pair*. They have distinct advantages in the sense that the number of processes and expedited situations corresponding to each pair is different. Interestingly, when compared with the previous algorithms, our algorithm is optimal in terms of the number of processes required for one-step decision. For example, frequency-based pair instantiation of our algorithm breaks the lower bound proved by BOSCO [66] for one-step Byzantine consensus. Table 1.4.1 compares the frequency-based pair instantiation of our algorithm with the previous results.

	System Model	**Failure Model**	**Number of Processes**	**Situations guaranteeing One-step Decision**	**Situations guaranteeing Two-step Decision**
Brasileiro et al. [14]	Asyn.	Crash	3t+1	Agreed proposals	–
Mostefaoui et al. [56]	Syn.	Crash	t+1	Condition-Based	–
Izumi et al. [49]	Asyn.	Crash	3t+1	Condition-Based	–
Friedman et al. [42]	Asyn.	Byzan.	7t+ 1	Agreed proposals	–
Song et al. [66] (Bosco)	Asyn.	Byzan.	5t+1 (Weak)	Agreed proposals	–
			7t+ 1 (Strong)	Agreed proposals of correct processes	–
Our Results (DEX)	**Asyn.**	**Byzan.**	**6t+1**	**Condition-Based**	**Condition-Based**

Table 1.2: Performance comparison of DEX with existing works.

- One drawback of the proposed framework is that it trades the decision scheme at third step for double-expedition property. This drawback causes a performance degradation in consensus-based applications when we consider pessimistic runs (that is, when the given input is out of the conditions). However, standing on its optimistic counterpart, we make a large number of inputs belong to the conditions so that our algorithm decides in one or two steps for many cases and achieves better performance in average.

To the best of our knowledge, double expedition property is the concept newly introduced by us. Hence, our work is the first result showing the feasibility for both one- and two-step decisions simultaneously with no help of additional stronger assumptions.

1.4.2 Improved Resilience against Mobile Byzantine Faults

Next, we study mobile Byzantine consensus in synchronous systems in the presence of stronger malicious agents, modelled by Garay[43]. We name these agents as *Code-Erasable Free-moving (CEF) agents*, as they can move with full speed at any time during a round and make corrupted processes forget that they run the algorithm by erasing their local memories (as a result, the recovering processes must learn the current state of computation including the code from currently correct processes before participating in the on-going execution).

We present a mobile Byzantine Consensus algorithm *MBC* that outperforms Garay's algorithm in the same settings. That is, our algorithm requires $n > 4t$ instead of $n > 6t$ (Garay's result) to tolerate *CEF* agents. In addition, we introduce a new model of agents, named *Code-Intact Free-moving (CIF) agents*, which do move with full speed at any time during a round, but do not erase the local memories of corrupted processes. In contrast to Garay's model, we allow processes to re-join the on-going execution with some predefined state. We show that our algorithm *MBC* also works with this model with minor modifications. Note that, this new model allows the recovering processes to rejoin the on-going execution without learning the current state

of computation while Garay's model requires that processes must learn it before reintegration.

Table 1.3 compares our results with the existing ones. The circled numbers can be described as follows:
①: an agent can erase the local memory of a process.
②: an agent can take a movement at any time during a round.
③: an agent can corrupt a new process before currently corrupted process recovers.
④: Processes can re-join the execution with the current state of computation.
⑤: Processes can re-join the execution with some predefined state.

	CEF Agent model ① + ② + ③ + ④	**CIF Agent model** ② + ③ + ⑤	**Weak agent model** ① + ④
Garay [43]	$n > 6t$	–	–
Burhman et al. [15]	–	–	$n > 3t$
Our Results (MBC)	**n > 4t**	**n > 4t**	–

Table 1.3: Performance comparison of MBC with previous works

1.5 Organization

The thesis consists of four chapters. In chapter 2, we introduce a new concept, called double-expedition property, for one-step Byzantine consensus and present an adaptive doubly-expedited one-step Byzantine consensus algorithm (*DEX*) for asynchronous systems. In chapter 3, we study mobile consensus problem in synchronous systems and present an efficient mobile Byzantine consensus algorithm (*MBC*) that provides better resilience than the existing algorithms. In Chapter 4, we present the summary of our results and future directions for pursuing research in Byzantine consensus.

Chapter 2

Adaptive and Doubly-Expedited One-Step Byz. Consensus

One of the important optimization aspects of consensus algorithms is to expedite the decision in favorable situations. For instance, assuming $n > 3t$, algorithms achieve consensus in one communication step when all processes propose the same value. In the literature, such algorithms are called *one-step consensus algorithms*. In this chapter, we investigate one-step consensus in asynchronous systems in the presence of an adaptive, threshold, message-conscious Byzantine adversary having unbounded computing power. In general, one-step algorithms are optimal (i.e., one-step) for very few inputs and no one-step algorithm supports a two-step decision. We present a novel adaptive doubly-expedited one-step algorithm, called *DEX*, that circumvents these difficulties using adaptive condition-based approach [48]. In this approach, a condition is a subset of all possible input vectors (whose entries correspond to the proposal of each process) and the contents of the condition dynamically changes according to the actual number of faulty processes (typically, a small number of faults allow the condition with a large number of inputs). The adaptiveness property is defined by using a sequence of conditions, called condition sequence. When DEX is instantiated with a pair of such condition sequences (each corresponds to one-step and two-step

decisions respectively), it gets the ability to provide both one-step and two-step decisions for a large number of inputs when there are fewer failures. We define some legality criteria for condition-sequence pairs and show that DEX can be instantiated with any condition-sequence pair satisfying them. We also present two examples for such legal condition-sequence pairs, namely *frequency-based pair* and *privileged-value-based pair*.

This chapter is organized as follows: Section 2.1 presents the system model, the definition of the Byzantine consensus problem and other necessary formalizations. Section 2.2 provides the legality criteria for doubly-expedited condition-sequence pairs and two examples of condition-sequence pairs satisfying them. Section 2.3 describes the generic framework of the doubly-expedited one-step Byzantine consensus algorithm DEX and proves its correctness. Section 2.4 presents a discussion on our algorithm DEX and some interesting problems for future work.

2.1 Preliminaries

2.1.1 System Model

We assume an asynchronous distributed system that consists of a set $\prod = \{p_1, p_2, \ldots, p_n\}$ of n processes. Each process communicates with each other process by sending messages over a reliable link where message loss, duplication and corruption never occur. Besides, there is no assumption about the relative speed of processes or about the timely delivery of messages.

The adversary that we consider is an adaptive, threshold, message-conscious Byzantine adversary having unbounded computing power. By this, we mean that the adversary can corrupt at most t processes out of the n processes during the algorithm execution, and it controls and coordinates the actions of the corrupted/faulty processes. Also, since it has full knowledge of the contents of messages, for example, it may choose to arbitrarily delay messages between any two correct processes (but messages are eventually delivered). A faulty process can behave arbitrarily, which means that it is allowed even not to follow the deployed algorithm. A process that is not faulty is said to be *correct*. Each process knows the value of t in advance. We also denote by

k, the actual number of failures during executions. Notice that, no process can be aware of the value of k. Throughout this chapter, we assume $n > 3t$, which is the necessary assumption to solve Byzantine consensus [51, 59].

2.1.2 Problem Definition

In Byzantine consensus problem, each correct process p_i has an initial value v_i from the set $\mathcal{V}$ of all possible initial values, and decides a value v according to the following rules:

- **Termination:** Each correct process eventually decides a value.
- **Agreement:** If two correct processes decide, they must decide the same value.
- **Unanimity:** If all correct processes propose the same value v, then no correct process decides a value different from v.

2.1.3 Underlying Consensus Primitive

In general, the consensus problem can not be solved in asynchronous systems without any additional assumptions [39]. Hence, we need some assumptions to guarantee the correct termination for arbitrary inputs. Many kinds of assumptions, such as partial synchrony, failure detectors, etc., are considered in past literatures. As our research direction is finding the feasibility of one-step decision, we simply assume an abstraction of them. More precisely, the system is assumed to be equipped with an *underlying consensus primitive* that ensures agreement, termination and unanimity, but provides no guarantees about its running time.

2.1.4 Adaptive Condition-Based Approach

The condition-based approach is one of the sophisticated methods used to overcome several impossibility results in the consensus problem (e.g., impossibility of consensus in asynchronous systems, time complexity lower bounds in synchronous consensus). The principle of this approach is to restrict inputs so that the generally-unsolvable problem can become solvable. In this

approach, an input vector is an n-dimensional vector, whose i-th entry contains a value proposed by process p_i. Note that, since a faulty process can propose different values to distinct processes, the entries corresponding to Byzantine processes are regarded to contain meaningless values. A *condition* defined for n processes is a subset of all possible input vectors. Adaptiveness in the condition-based approach is a property that allows a condition to change dynamically according to the actual number of faulty processes. That is, the fewer failures allow the condition with a large number of input vectors. Adaptiveness of a condition is defined by a *condition sequence* $(C_0, C_1, ..., C_k, ..., C_t)$ satisfying $C_k \supseteq C_{k+1}$ for any $k(0 \leq k \leq t-1)$, where the k-th condition corresponds to the set of input vectors that is valid when the actual number of faults is less than or equal to k.

2.1.5 Doubly-Expedited Consensus

In this subsection, we introduce a novel feature of consensus algorithms, called *double-expedition* property. In the execution of a doubly-expedited algorithm, each process has two chances for fast decision, i.e., it can decide in either one step or two steps. Since the fast decision is guaranteed only for good inputs, we introduce the condition-sequence pair as follows: $(S^1, S^2) = ((C_0^1, C_1^1, \cdots, C_k^1, \cdots, C_t^1), (C_0^2, C_1^2, \cdots, C_k^2, \cdots, C_t^2))$, where S^1 and S^2 correspond to the condition sequences that identify the situations guaranteeing one-step and two-step decisions respectively such that for any $k(0 \leq k \leq t)$, $C_k^1 \subset C_k^2$. For example, consider the input vector I such that $I \notin C_k^1$, $I \in C_{k-1}^1$ and $I \in C_k^2$ hold. Then, if I is given to the consensus algorithm and the actual number of faulty processes is less than k, all processes decide in one step because $I \in C_{k-1}^1$. If (exactly) k processes are faulty, then one-step decision is no more guaranteed, but all the processes necessarily decide in two steps because of $I \in C_k^2$.

2.2 Legality for Double Expedition

It is clear that we cannot design doubly-expedited consensus algorithms for any pair of condition sequences. In this section, we propose sufficient crite-

ria such that we can construct a doubly-expedited algorithm characterized by any condition-sequence pair satisfying them. In addition, we show two examples of condition-sequence pairs satisfying these criteria.

2.2.1 Notations

Let $\mathcal{V}$ be an ordered set of all possible proposal values. We introduce the default value $\perp$ not in $\mathcal{V}$. Let I be an input vector in $\mathcal{V}^n$. We define a *view* J of I to be a vector in $(\mathcal{V} \cup \{\perp\})^n$, which is obtained by replacing at most t entries in I by $\perp$. We denote $J[k]$ as the k-th element of J. The number of occurrences of a value v in a view J is denoted by $\#_v(J)$. For two views J_1 and J_2, the containment relation $J_1 \leq J_2$ is defined as $\forall k(1 \leq k \leq n) : J_1[k] \neq \perp \Rightarrow J_1[k] = J_2[k]$. Also, for two views J_1 and J_2, let $dist(J_1, J_2)$ be the Hamming distance between J_1 and J_2 (that is, $dist(J_1, J_2) = |\{k \in \{1, 2, ..n\} | J_1[k] \neq J_2[k]\}|$). As well as, $\mathcal{V}_k^n$ denotes the set of all views where the $\perp$ value appears at most k times. The number of non-default values in J is denoted by $|J|$.

We show the two fundamental properties holding for input vectors and their views.

Proposition 1 Given an input vector I and any proposed value $a \in \mathcal{V}$, if $\exists I' : dist(I, I') \leq k$, then $\#_a(I') \geq \#_a(I) - k$ and $\#_a(I') \leq \#_a(I) + k$ hold.

Lemma 1 Given any two views J and J' and any proposed value $a \in \mathcal{V}$, if $\exists I, I' : J \leq I \wedge J' \leq I' \wedge dist(I, I') \leq k$, the following inequalities hold.

- $\#_a(J') \geq \#_a(J) - \#_\perp(J') - k$.
- $\#_a(J') \leq \#_a(J) + \#_\perp(J) + k$.

Proof Consider two views J and J' such that $J \leq I \wedge J' \leq I' \wedge dist(I, I') \leq k$. Let x and y be the number of entries that contain $\perp$ in J and J', respectively. Since $J \leq I$, at most x entries of I, corresponding to $\perp$'s in J, can contain a. Hence, we get $\#_a(I) \geq \#_a(J)$ and $\#_a(I) \leq \#_a(J)+x$. In addition, since $dist(I, I') \leq k$, from proposition 1, it follows that $\#_a(I') \geq \#_a(I) - k$ and $\#_a(I') \leq \#_a(I)+k$. From the above inequalities, it is clear that $\#_a(I') \geq$

$\#_a(J) - k$ and $\#_a(I') \leq \#_a(J) + x + k$. Then, as $J' \leq I'$, at most y entries of J', corresponding to a's in I', can contain $\perp$. It implies that $\#_a(J') \leq \#_a(I')$ and $\#_a(J') \geq \#_a(I') - y$. Hence, we get $\#_a(J') \geq \#_a(J) - y - k$ and $\#_a(J') \leq \#_a(J) + x + k$. Thus, the lemma holds. □

2.2.2 Legality Criteria

Given a pair of condition sequences (S^1, S^2), we consider two predicates $P1, P2 : \mathcal{V}_t^n \rightarrow \{\text{True}, \text{False}\}$[1] and a function $F : \mathcal{V}_t^n \rightarrow \mathcal{V}$. Then, (S^1, S^2) is said to be *legal* if we can define $P1$, $P2$ and F satisfying the following five properties:

- (LT1) $\forall J \in \mathcal{V}_t^n : \exists I : I \in C_k^1 \wedge dist(J, I) \leq k \Rightarrow P1(J)$.
- (LT2) $\forall J \in \mathcal{V}_t^n : \exists I : I \in C_k^2 \wedge dist(J, I) \leq k \Rightarrow P2(J)$.
- (LA3) $\forall J, J' \in \mathcal{V}_t^n : P1(J) \wedge \exists I, I' : J \leq I \wedge J' \leq I' \wedge dist(I, I') \leq t \Rightarrow F(J) = F(J')$.
- (LA4) $\forall J, J' \in \mathcal{V}_t^n : P2(J) \wedge \exists I : J \leq I \wedge J' \leq I \Rightarrow F(J) = F(J')$.
- (LU5) $\forall J \in \mathcal{V}_t^n : F(J) = a \Rightarrow \#_a(J) > t \vee a$ is the most common non-$\perp$ value in J.

These properties are used to enforce the basic requirements of the doubly-expedited Byzantine consensus. Informally, $P1$ and $P2$ are the predicates to test whether the current view of a process contains sufficient information to decide in one or two step(s) respectively, and F is the function to obtain the decision value from the current view. The first property *LT1* is used for imposing one-step termination. The predicate *P1* must allow each correct process to decide in one step if its view has the possibility to come from an input vector included in the condition C_k^1 and the actual number of failures is less than or equal to k. Similarly, the property *LT2* corresponds to two-step decision. The property *LA3* states that if a process decides in one step based on its view J, using F, then at any process with view J', the

[1] In what follows $P1(J) = \text{true}$ is abbreviated as $P1(J)$

equality F(J')=F(J) holds, provided P1(J) holds. The property *LT4* works in a similar manner for two-step decision. More precisely, the property *LA3* (or *LA4*) enforces the agreement between one-step (or two-step) decision and others. The last property *LU5* is the one to guarantee unanimity.

2.2.3 Examples for Legal Condition-Sequence Pairs

2.2.3.1 Frequency-Based Pair

Here we introduce a legal condition-sequence pair (P^{freq}) based on the frequency-based condition given in [55], and prove its legality. Let $1st(J)$ be a non-$\perp$ value that appears most often in a vector J. If two or more values appear most often in J, then the largest one is selected. Let $\hat{J}$ be a vector obtained by replacing $1st(J)$ from J by $\perp$, and we define $2nd(J) = 1st(\hat{J})$. That is, $2nd(J)$ is the second most frequent value in J. The frequency-based condition C_d^{freq} is defined as follows:

$$C_d^{freq} = \{I \in \mathcal{V}^n | \#_{1st(I)}(I) - \#_{2nd(I)}(I) > d\}$$

It is known that C_d^{freq} belongs to *d-legal* conditions [55]., which is the class of conditions that are necessary and sufficient to solve the consensus in failure prone asynchronous systems, where at most d processes can crash. Using this condition, we can construct the frequency-based condition-sequence pair (P^{freq}) with the associated parameters $P1^{freq}$, $P2^{freq}$ and F^{freq} as follows:

$P^{freq} = (S^1, S^2) = ((C_0^1, C_1^1, C_2^1, \cdots, C_k^1, \cdots, C_t^1), (C_0^2, C_1^2, C_2^2, \cdots, C_k^2, \cdots, C_t^2))$
where $C_k^1 = C_{4t+2k}^{freq}$ and $C_k^2 = C_{2t+2k}^{freq}$

- $P1^{freq}(J) \equiv \#_{1st(J)}(J) - \#_{2nd(J)}(J) > 4t$.
- $P2^{freq}(J) \equiv \#_{1st(J)}(J) - \#_{2nd(J)}(J) > 2t$.
- $F^{freq}(J) = 1st(J)$.

Notice that, a stronger assumption $n > 6t$ is required to construct P^{freq}.

Theorem 1 The condition-sequence pair P^{freq} is legal.

Proof ***LT1:*** We have to show that $\forall J \in \mathcal{V}_t^n : \exists I \in C_k^1 \wedge dist(J,I) \leq k \Rightarrow P1(J)$. That is, $\#_{1st(I)}(I) - \#_{2nd(I)}(I) > 4t + 2k \wedge dist(J,I) \leq k \Rightarrow \#_{1st(J)}(J) - \#_{2nd(J)}(J) > 4t$.

Since $I \in C_k^1$, we have $\#_{1st(I)}(I) - \#_{2nd(I)}(I) > 4t + 2k$. As $dist(J,I) \leq k$, from proposition 1, it follows that $\#_{1st(I)}(J) \geq \#_{1st(I)}(I) - k$ and $\#_x(J) \leq \#_x(I) + k$ for any value $x \neq 1st(I)$. Since $2nd(I)$ is the second most frequent value in I, $\#_x(J) \leq \#_{2nd(I)}(I) + k$. Hence, $\#_{1st(I)}(J) - \#_x(J) \geq \#_{1st(I)}(I) - k - \#_{2nd(I)}(I) - k$. Therefore, we get $\#_{1st(I)}(J) - \#_x(J) > 4t$. It implies that $1st(I) = 1st(J)$ and $\#_{1st(J)}(J) - \#_{2nd(J)}(J) > 4t$.

LT2: We have to show that $\forall J \in \mathcal{V}_t^n : \exists I \in C_k^2 \wedge dist(J,I) \leq k \Rightarrow P2(J)$. That is, $\#_{1st(I)}(I) - \#_{2nd(I)}(I) > 2t + 2k \wedge dist(J,I) \leq k \Rightarrow \#_{1st(J)}(J) - \#_{2nd(J)}(J) > 2t$.

The proof is almost the same as the proof of *LT1* (with only replacing C_k^1 and $4t$ with C_k^2 and $2t$ respectively).

LA3: Consider $J, J' \in \mathcal{V}_t^n$. We have to show that if $P1^{freq}(J) \wedge \exists I, I' : J \leq I \wedge J' \leq I' \wedge dist(I,I') \leq t$, then $1st(J) = 1st(J')$.

Since $P1^{freq}(J)$ holds, $\#_{1st(J)}(J) - \#_v(J) > 4t$ also holds for any value $v \neq 1st(J)$. Then, as $J, J' \in \mathcal{V}_t^n \wedge J \leq I \wedge J' \leq I' \wedge dist(I,I') \leq t$, from lemma 1, it follows that $\#_{1st(J)}(J') \geq \#_{1st(J)}(J) - 2t$ and $\#_v(J') \leq \#_v(J) + 2t$. Hence, we get $\#_{1st(J)}(J') - \#_v(J') \geq \#_{1st(J)}(J) - \#_v(J) - 4t$. Since $\#_{1st(J)}(J) - \#_v(J) > 4t$, it is clear that $\#_{1st(J)}(J') - \#_v(J') > 0$. Thus, we get $1st(J) = 1st(J')$.

LA4: Consider $J, J' \in \mathcal{V}_t^n$. It suffices to show that if $P2^{freq}(J) \wedge \exists I : J \leq I \wedge J' \leq I$, then $1st(J) = 1st(J')$.

Since $P2^{freq}(J)$ holds, $\#_{1st(J)}(J) - \#_v(J) > 2t$ also holds for any $v \neq 1st(J)$. Then, as $J, J' \in \mathcal{V}_t^n \wedge J \leq I \wedge J' \leq I \wedge dist(I,I) = 0$, from lemma 1, it follows that $\#_{1st(J)}(J') \geq \#_{1st(J)}(J) - t$ and $\#_v(J') \leq \#_v(J) + t$. Hence, we get $\#_{1st(J)}(J') - \#_v(J') \geq \#_{1st(J)}(J) - \#_v(J) - 2t$. Since $\#_{1st(J)}(J) - \#_v(J) > 2t$, it follows that $\#_{1st(J)}(J') - \#_v(J') > 0$. Therefore, we conclude that $1st(J) = 1st(J')$ holds.

LU5: This property is trivially satisfied since $1st(J)$ is the most frequent non-$\perp$ value in J. $\square$

2.2.3.2 Privileged-Value-Based Pair

Here, we present another legal condition-sequence pair (P^{prv}), constructed from a privileged-value-based condition, and prove its legality. In some practical agreement problems such as atomic commitment, a single value (e.g., commit) is often proposed by most of the processes. The previous results [14, 45] have shown that, if this value is assigned some privilege, it is possible to expedite the decision. Let us assume that there is a value (say m) that is privileged among the set of all proposal values. Each process knows the value m a priori. Then, the privileged-value-based condition $C_d^{prv(m)}$ can be defined as follows:

$$C_d^{prv(m)} = \{I \in \mathcal{V}^n | \#_m(I) > d\}$$

Note that $C_d^{prv(m)}$ also belongs to *d-legal* conditions [55]. Using this condition, we can construct the privileged-value-based condition-sequence pair P^{prv} with the related parameters $P1^{prv}$, $P2^{prv}$ and F^{prv} as follows:

$P^{prv} = (S^1, S^2) = ((C_0^1, C_1^1, C_2^1, \cdots, C_k^1, \cdots, C_t^1), (C_0^2, C_1^2, C_2^2, \cdots, C_k^2, \cdots, C_t^2))$
where $C_k^1 = C_{3t+k}^{prv(m)}$ and $C_k^2 = C_{2t+k}^{prv(m)}$

- $P1^{prv}(J) \equiv \#_m(J) > 3t$.
- $P2^{prv}(J) \equiv \#_m(J) > 2t$.
- $F^{prv}(J) = m$ if $\#_m(J) > t$. Otherwise, $F^{prv}(J) =$ the most frequent non-$\perp$ value in J.

Notice that, the assumption $n > 4t$ is required to make P^{prv} meaningful.

Theorem 2 The condition-sequence pair P^{prv} is legal.

Proof ***LT1:*** Assume $I \in C_k^1$. We have to show that $\forall J \in \mathcal{V}_t^n$, if $dist(J, I) \leq k$, then $\#_m(J) > 3t$ holds.

Since $I \in C_k^1$, $\#_m(I) > 3t + k$ holds. Then, as $dist(J, I) \leq k$, it follows that $\#_m(J) \geq \#_m(I) - k$. Hence, $\#_m(J) > 3t + k - k$. This implies that $\#_m(J) > 3t$.

LT2: Assume $I \in C_k^2$. We have to show that $\forall J \in \mathcal{V}_t^n$, if $dist(J, I) \leq k$, then $\#_m(J) > 2t$ holds.

This proof is almost the same as the proof of *LT1* (with only replacing C_k^1 and $3t$ with C_k^2 and $2t$ respectively).

LA3: Consider $J, J' \in \mathcal{V}_t^n$. We have to show that if $P1^{prv}(J) \wedge \exists I, I' : J \leq I \wedge J' \leq I' \wedge dist(I, I') \leq t$, then $F^{prv}(J) = F^{prv}(J')$.

Since $P1^{prv}(J)$ holds, $\#_m(J) > 3t$ and $F^{prv}(J) = m$ also hold. Then, as $J, J' \in \mathcal{V}_t^n \wedge J \leq I \wedge J' \leq I' \wedge dist(I, I') \leq t$, from lemma 1, it is directly implied that $\#_m(J') \geq \#_m(J) - 2t$. Since $\#_m(J) > 3t$, it follows that $\#_m(J') > t$. Hence, $F^{prv}(J') = m = F^{prv}(J)$.

LA4: Consider $J, J' \in \mathcal{V}_t^n$. We have to show that if $P2^{prv}(J) \wedge \exists I : J \leq I \wedge J' \leq I$, then $F^{prv}(J) = F^{prv}(J')$.

Since $P2^{prv}(J)$ holds, it follows that $\#_m(J) > 2t$ and $F^{prv}(J) = m$. Also, as $J, J' \in \mathcal{V}_t^n \wedge J \leq I \wedge J' \leq I \wedge dist(I, I) = 0$, from lemma 1, it is implied that $\#_m(J') \geq \#_m(J) - t$. Since $\#_m(J) > 2t$, we get $\#_m(J') > t$. Hence, we conclude that $F^{prv}(J') = m = F^{prv}(J)$.

LU5: This property is trivially satisfied because $F^{prv}(J)$ is either m (when $\#_m(J) > t$) or the most frequent non-$\perp$ value in J. □

2.3 Adaptive and Doubly-Expedited Algorithm

This section presents a generic doubly-expedited algorithm *DEX* for one-step Byzantine consensus that can be instantiated with any legal condition-sequence pair.

Function *Consensus*(v_i)

Initially $J1_i[k] = \perp$, $J2_i[k] = \perp$ and $echo\text{-}sent_i[k]$ = **False** for each $k, 1 \leq k \leq n$;
$decided_i$ = **False**, $proposed_i$ = **False**, $noc_i = 0$

begin

1 : **Upon** $\mathsf{Propose_i}(v_i)$ **do:**
2 : $\mathsf{Broadcast_i}(PROP, v_i)$;

3 : **Upon** $\mathsf{Receive_i}(PROP, v_j)$ from any process p_j **do:**
4 : **if** $echo\text{-}sent_i[j]$ = **False then**
5 : $J1_i[j] = v_j$;
6 : $\mathsf{Broadcast_i}(ECHO, v_j, j)$;
7 : $echo\text{-}sent_i[j]$ = **True**;
8 : **end if**
9 : **if** $|J1_i| \geq n - t$ and $\mathsf{P1}(J1_i)$ and $decided_i$ = **False then**
10 : $\mathsf{Decide_i}(F(J1_i))$; $decided_i$ = **True**;
11 : **end if**

12 : **Upon** $\mathsf{Receive_i}(ECHO, v_j, j)$ **do:**
13 : noc_i = number of copies of $(ECHO, v_j, j)$
14 : received so far from distinct processes;
15 : **if** $noc_i > (n + t)/2$ **then** $J2_i[j] = v_j$; **end if**
% Hence, $((J2_i[k] \neq \perp) \wedge (J2_j[k] \neq \perp)) \Rightarrow (J2_i[k] = J2_j[k])$%
16 : **if** $|J2_i| \geq n - t$ and $proposed_i$ = **False then**
17 : $\mathsf{UC\_propose_i}(F(J2_i))$;
18 : $proposed_i$ = **True**;
19 : **end if**
20 : **if** $|J2_i| \geq n - t$ and $\mathsf{P2}(J2_i)$ and $decided_i$ = **False then**
21 : $\mathsf{Decide_i}(F(J2_i))$; $decided_i$ = **True**;
22 : **end if**

23 : **Upon** $\mathsf{UC\_decide_i}(v)$ **do:**
24 : **if** $decided_i$ = **False then**
25 : $\mathsf{Decide_i}(v)$; $decided_i$ = **True**;
26 : **end if**

end

Figure 2.1: Algorithm *DEX*

2.3.1 Algorithm *DEX*

Figure 2.1 provides the pseudocode of the algorithm. Our algorithm provides each process p_i with two events $\mathsf{Propose_i}(v)$ and $\mathsf{Decide_i}(v')$. p_i initially proposes a value v using $\mathsf{Propose_i}(v)$ and eventually decides a value v' using $\mathsf{Decide_i}(v')$. The processes exchange messages using the standard Broadcast/Receive primitives. The underlying consensus is served by two primitives $\mathsf{UC\_propose_i}(v)$ and $\mathsf{UC\_decide_i}(v)$ which correspond to proposal of a value v and decision by v respectively.

For two-step decision, our algorithm actually exploits an idea in [3] used for simulating *Identical Byzantine* failure model on the top of general Byzantine failure model. Informally, the Identical Byzantine system guarantees that even if Byzantine processes send arbitrary messages, all processes that receive a message from a faulty process receive the same message. Figure 2.2 shows how the Identical Byzantine system works. In this system, a correct process accepts a message only when it knows there are enough witnesses for the broadcast of this message. More precisely, in identical Byzantine failure model, Byzantine failures appear as crash failures.

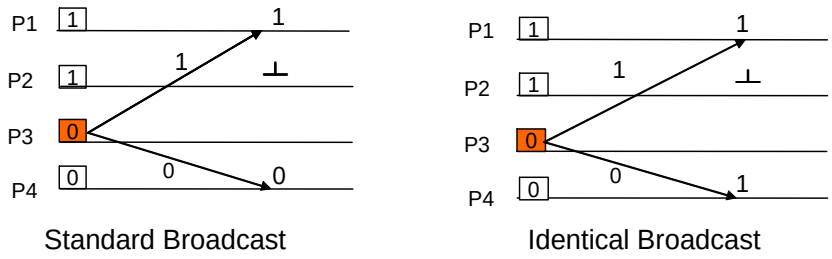

Figure 2.2: Identical Byzantine system: Let processes $P1, P2, P4$ are correct and $P3$ is faulty; Even if $P3$ sends different messages to $P1$ and $P4$, they receive the same message.

Our algorithm works as follows: Each process p_i starts a consensus execution with the invocation of $Consensus(v_i)$ where v_i is its initial proposal value. First, each process p_i broadcasts a message $< PROP, v_i >$ to provide the other processes with v_i. When p_i receives $< PROP, v_j >$ from any

process p_j, it stores v_j into the view $J1_i$, which corresponds to one-step decision. Also, it acts as a witness for that broadcast and sends its own message $< ECHO, v_j, j >$ to all processes unless it has already echoed for process p_j. Then, when at least $n-t$ messages are received at $J1_i$, p_i tries to make a decision by evaluating $P1(J1_i)$. If $P1(J1_i)$ is true, p_i immediately decides $F(J1_i)$, that is, it decides in one step. Otherwise, p_i repeats the attempt whenever $J1_i$ is updated. When p_i receives the same message $< ECHO, v_j, j >$ from more than $(n+t)/2$ distinct processes, it accepts that message and stores v_j into the view $J2_i$, which corresponds to two-step decision. When p_i receives at least $n-t$ messages at $J2_i$, it activates the underlying consensus with $F(J2_i)$. In addition, it evaluates $P2(J2_i)$ to check whether $J2_i$ is sufficient for taking decision. If $P2(J2_i)$ is true, p_i immediately decides $F(J2_i)$, that is, it decides in two steps. Otherwise, p_i repeats the check with each update at $J2_i$. Also, when the underlying consensus decides, each p_i simply borrows the decision of the underlying consensus unless it has decided already.

2.3.2 Correctness

We prove the correctness of our algorithm by showing that it provides both one- and two-step decisions when it is instantiated with any legal condition-sequence pair (S^1, S^2). In the following proofs, let I be the actual input vector, and I_i^1, I_i^2 be vectors obtained respectively from the views $J1_i, J2_i \in \mathcal{V}_t^n$ by replacing the default values with corresponding values in I. Remember that, since we know that the Byzantine problem cannot be solved with $t \geq n/3$, we assume that $n > 3t$.

Lemma 2 Let p_i and p_j be two correct processes. If $(J2_i[k] \neq \perp) \wedge (J2_j[k] \neq \perp$), then $J2_i[k] = J2_j[k]$ for each $k (1 \leq k \leq n)$.

Proof It is proved by contradiction. Let us consider two correct processes p_i and p_j such that $J2_i[k] = v$, $J2_j[k] = v'$ and $v \neq v'$ for some process p_k. It follows that p_i and p_j have collected echo messages from more than $(n+t)/2$ distinct processes for v and v' each. Notice that, any two sets of $(n+t)/2$ processes have at least $t+1$ processes in common. Since there are

only t Byzantine processes, at least one correct process must have sent two different echo messages for p_k. But, it follows directly from the algorithm code that, for any process k, a correct process will send the same echo message to all processes. This is a contradiction. □

Lemma 3 For any two correct processes p_i and p_j, (1) if $J1_i[k] = v$ and/or $J2_j[k] = v$, then v is proposed by process p_k, and (2) if $(J1_i[k] \neq \perp) \wedge (J2_j[k] \neq \perp)$, then $J1_i[k] = J2_j[k]$ for each correct process p_k, $(1 \leq k \leq n)$.

Proof Assume that $J1_i[k] = v$, $J2_j[k] = v'$ and $v \neq v'$ for some correct process p_k. Since links between processes are private, a Byzantine process cannot forge a message from a correct process. Hence, it is clear that if $J1_i[k] = v$, then v is proposed by p_k. Then, from $J2_j[k] = v'$, it is implied that p_j must have collected the same echo message $< ECHO, v', k >$ from more than $(n + t)/2$ distinct processes. Notice that, a correct process broadcasts an echo message $< ECHO, v', k >$ for p_k if and only if it receives a proposal message $< PROP, v' >$ from p_k. Also, since there are only t Byzantine processes such that $t < (n + t)/2$, they cannot make p_j accept for p_k a value that is not proposed by p_k. Therefore, it is sure that v' is proposed by p_k. Then, if $v \neq v'$, it follows that the correct process p_k have sent two different proposal values. This is a contradiction. □

Lemma 4 (Termination) Each correct process p_i eventually decides.

Proof Since there are at most t Byzantine processes, each correct process p_i receives messages from at least $n - t$ processes. It implies that, at some point $|J2_i| \geq n - t$. Hence, p_i certainly initiates the underlying consensus. Since the underlying consensus guarantees termination, p_i can decide when the underlying consensus decides. It follows that each process eventually decides. □

Lemma 5 (Agreement) No two correct processes decide different values.

Proof Let two correct processes p_i and p_j decide v_i and v_j respectively. Then, we prove that $v_i = v_j$. Consider the following six cases.

- (**Case 1**) *When both p_i and p_j decide in one step at line 10.*

  Since both p_i and p_j decide in one step, $P1(J1_i)$ and $P1(J1_j)$ hold. Let us construct two vectors I_i^1 and I_j^1. From the definitions of I_i^1 and I_j^1, it follows that $J1_i \leq I_i^1$ and $J1_j \leq I_j^1$ hold. Since there are at most t Byzantine processes, and only the Byzantine processes send different values to distinct processes, the vectors I_i^1 and I_j^1 can differ in at most t entries . Hence, $dist(I_i^1, I_j^1) \leq t$ also holds. From property *LA3*, it is clear that $v_i = F(J1_i) = F(J1_j) = v_j$. Thus, we can conclude that p_i and p_j decide the same value.

- (**Case 2**) *When p_i decides in one step at line 10 and p_j decides in two steps at line 21.*

  Since p_i and p_j decide in one and two step(s) respectively, $P1(J1_i)$ and $P2(J2_j)$ hold. Similar to Case 1, we can construct two vectors I_i^1 and I_j^2 such that $J1_i \leq I_i^1$ and $J2_j \leq I_j^2$ hold. From lemma 3, it follows that the vectors I_i^1 and I_j^2 can differ in at most t entries correspond to Byzantine processes. Hence, $dist(I_i^1, I_j^2) \leq t$ also holds. From property *LA3*, it is clear that $v_i = F(J1_i) = F(J2_j) = v_j$.

- (**Case 3**) *When both p_i and p_j decide in two steps at line 21.*

  Since p_i and p_j decide in two steps, $P2(J2_i)$ and $P2(J2_j)$ hold. From lemma 2, it follows that if an entry in $J2_i$ contains a non-$\perp$ value v, then the same entry in $J2_j$ contains either v or $\perp$ and vice versa. Hence, it is possible to have a vector I' such that $\forall k(1 \leq k \leq n) : (J2_i[k] \neq \perp \Rightarrow I'[k] = J2_i[k]) \wedge (J2_j[k] \neq \perp \Rightarrow I'[k] = J2_j[k])$. This implies that $J2_i \leq I'$ and $J2_j \leq I'$ hold. Thus, from property $LA4$, we get $v_i = F(J2_i) = F(J2_j) = v_j$.

- (**Case 4**) *When p_i decides in one step at line 10 and p_j decides using underlying consensus at line 25.*

  Since p_j decides using the underlying consensus, and the underlying consensus satisfies unanimity, it is sufficient to show that every correct process p_k proposes v_i at line 17. We know that p_i decides using $J1_i$ and p_k uses $J2_k$ to propose a value to the underlying consensus. Construct

the two vectors I_i^1 and I_k^2. By the same argument as Case 2, we can show that $J1_i \leq I_i^1$, $J2_k \leq I_k^2$ and $dist(I_i^1, I_k^2) \leq t$ hold. From property *LA3*, we get $v_i = F(J1_i) = F(J2_k) = v_k$. Hence, it is clear that each process p_k proposes v_i.

- (**Case 5**) *When p_i decides in two steps at line 21 and p_j decides using underlying consensus at line 25.*

  Since p_j decides by the underlying consensus, similar to Case 4, we have to show that every correct process p_k proposes v_i to the underlying consensus at line 17. We know that p_i decides using $J2_i$ and p_k uses $J2_k$ to propose a value to the underlying consensus. By the same argument as Case 3, we can construct a vector I' such that $J2_i \leq I'$ and $J2_k \leq I'$. From property $LA4$, it is clear that $v_i = F(J2_i) = F(J2_k) = v_k$. Hence, we can conclude that each process p_k proposes v_i.

- (**Case 6**) *When both p_i and p_j decide at line 25.*

  Since the underlying consensus guarantees agreement property, we can conclude that $v_i = v_j$.

□

Lemma 6 (Unanimity) If all correct processes propose the same value v, then no correct process decides a value different from v.

Proof Let k be the actual number of Byzantine processes and all correct processes propose the same value v. Since $k \leq t$, at each correct process p_i, no value except v appears more than t times in $J1_i$ and $J2_i$. If p_i decides at line 10 or 21, its decision value is either $F(J1_i)$ or $F(J2_i)$. From the definition of *LU5*, it follows that $F(J1_i) = F(J2_i) = v$. Hence, p_i decides v. In addition, since each p_i proposes $F(J2_i)$(that is, v) to the underlying consensus and the underlying consensus satisfies unanimity, any correct process that decides using underlying consensus decides only v. Hence, the unanimity holds. □

Lemma 7 The algorithm *DEX* guarantees one-step decision for any input vector I belonging to C_k^1 if at most k processes exhibit Byzantine behavior.

Proof Since there are at most k Byzantine processes, each correct process p_i is guaranteed to receive messages from $n - k$ correct processes. Hence, eventually $dist(J1_i, I) \leq k$ holds. From property $LT1$, it follows that p_i decides in one step. □

Lemma 8 The algorithm *DEX* guarantees two-step decision if the input vector I belonging to C_k^2 and at most k processes are Byzantine.

Proof As stated in lemma 7, since there are at most k Byzantine processes each correct process p_i can receive messages from all $n-k$ correct processes. Hence, eventually $dist(J2_i, I) \leq k$ holds. From property $LT2$, it is clear that p_i decides in two steps. □

The above lemmas imply the following theorem:

Theorem 3 For any instantiation with legal condition-sequence pairs, the algorithm *DEX* is a doubly-expedited one-step consensus algorithm.

2.4 Discussion and Open Problems

- When compared with a recent work BOSCO [66], DEX improves on the bound t, namely to be strong one-step it requires $n > 6t$ instead of $n > 7t$, and hence circumvents the lower bound ($n > 7t$) shown by BOSCO. This improvement is achieved from its adaptiveness property. More explicitly, previous Byzantine algorithms allow a process to wait for messages from only $n-t$ processes without the risk of being blocked forever. As a result, each process can miss messages from at most t correct processes. These algorithms in fact add t to the ratio between t and n to compensate the possible missing messages. However, since DEX allows each process to collect messages from all correct processes, such requirement is eliminated. This enables DEX to keep the number of processes n minimum. In addition, compared to BOSCO, DEX achieves stronger fast-decision guarantees, which can be explained by specifying BOSCO in the context of condition-based approach. The two variants of BOSCO, weak and strong ones, can be specified using

condition-sequence pairs as follows: $P = ((C_0^1, C_1^1, C_2^1, \cdots, C_k^1, \cdots, C_t^1), (C_0^2, C_1^2, C_2^2, \cdots, C_k^2, \cdots, C_t^2))$, where for the strong one-step $C_k^1 = C_{7t}^{freq}$ and $C_k^2 = \phi$ for each $k(0 \leq k \leq t)$, and for the weak one-step $C_0^1 = C_{5t}^{freq}$, $C_k^1 = \phi$ for each $k(1 \leq k \leq t)$ and $C_k^2 = \phi$ for each $k(0 \leq k \leq t)$. It shows that our frequency-based pair instantiation of DEX (where $C_k^1 = C_{4t+2k}^{freq}$ and $C_k^2 = C_{2t+2k}^{freq}$ for each $k(0 \leq k \leq t)$) is strictly better than BOSCO for the strong one-step. Also, for the weak one-step, we can construct a strictly better algorithm in our framework using a condition-sequence pair where $C_k^1 = C_{4t+2k}^{freq}$, for each $k(0 \leq k \leq \lfloor t/2 \rfloor), C_k^1 = \phi$ for each $k(\lfloor t/2 \rfloor < k \leq t)$ and $C_k^2 = C_{2t+2k}^{freq}$ for each $k(0 \leq k \leq t)$.

- Our algorithm *DEX* is optimally resilient for one-step Byzantine consensus and achieves better message complexity, that is $O(n^2)$. However, it does not meet the $n > 3t$ lower bound [51, 59] for Byzantine consensus in asynchronous systems. This can be viewed as the price to be paid to favor fast termination, simplicity and communication efficiency. (Code simplicity implies shorter development time and lower chances for bugs. Fast termination means better service to clients, and reduced communication complexity means improved throughput). As the cost of hardware goes down, it may be reasonable to prefer such solutions in some circumstances. For example, in order to tolerate one Byzantine failure, an algorithm that assumes $n > 3t$ but whose complexity is $O(n^3)$ messages, used in a system of 4 processes, generates in each round 64 messages. But, for algorithms (like ours) that assume $n > 4t$ or $n > 6t$, but with $O(n^2)$ messages per round, we get 25 and 49 messages, respectively. Moreover, the design of asynchronous Byzantine consensus that meet all lower bounds (on the value of t, the early decision in good circumstances, etc) is still an open problem.

- Unlike previous algorithms [19, 59], DEX does not require heavy mechanisms such as "message proofs", "certificates", or any kind of application level signatures. Also, DEX does not explicitly depend on any complicated mechanisms, such as partial synchrony and oracles, although the subroutine invoked by DEX may have such dependencies.

With a judicious choice of the consensus subroutine, DEX can tolerate a strong adaptive message-conscious adversary that can corrupt processes during the execution of the algorithm, arbitrarily re-order messages and collude with Byzantine processors. An example of an algorithm that can be used as a subroutine in DEX is the Ben-Or algorithm [4]. Algorithms that do not provide validity, such as PBFT [19], cannot be used by DEX.

We now conclude this chapter with a few interesting open questions:

- **Open Problem 1:** Can we eliminate the drawback of DEX? or we may ask: Can we design a one-step algorithm that does not waste any communication step?
- **Open Problem 2:** What is the lower bound on the number of processes for the strong one-step Byzantine consensus in asynchronous systems?
- **Open Problem 3:** Can we design a one-step algorithm with better message complexity than what is shown here?

Chapter 3

Improved Resiliency against Mobile Byzantine Faults

In this chapter, we study the problem of reaching and maintaining agreement among the set of non-faulty processes in the presence of a powerful adversary that distributes up to t malicious agents which can move from one process to another and try to corrupt them. This problem is referred to as mobile Byzantine consensus problem. In a previous result [43], Garay has shown that $n > 6t$ is sufficient to solve this problem even in the presence of strong agents that can move with full speed (in the sense that each agent can take a movement in every round) and can make corrupted processes forget that they run the algorithm (as a result, after recovery a process must learn the current state of computation including the code from other processes). In this chapter, we improve this result, by providing an algorithm *MBC* that requires $n > 4t$ to solve this problem in the same settings. In addition, we show that our algorithm can also work for a model in which any process can restart the algorithm immediately (but with some predefined state) after its recovery.

The organization of this chapter is as follows: Section 3.1 provides the system model, the definition of the mobile Byzantine consensus problem, and other necessary formalizations. In section 3.2, presents our mobile Byzantine consensus algorithm *MBC* and prove its correctness. Section 3.3 presents a discussion on our new algorithm and some interesting problems for future

work.

3.1 Preliminaries

3.1.1 System Model

We consider a distributed system that consists of n processes numbered from 1 to n. Each process communicates with each other by sending messages over a reliable link where neither message loss, duplication nor corruption occurs. Our system is synchronous. This means that its execution is organized by a sequence of rounds during which each process can send messages to other processes, receive messages, and perform some local computation. Also, a message sent in some round is necessarily received within in the same round.

We assume that the system is interfered by a powerful computationally unbounded adversary which can inject up to t malicious agents into the system. These agents can move from processes to processes at full speed and corrupt them in a dynamic fashion. Because of the mobility of agents, any process can be corrupted during the course of the algorithm. However, we assume that one process remains uncorrupted for $O(n)$ rounds, since a discouraging impossibility result [63] proved that consensus is not solvable without restrictions even with a single mobile failure. Since each agent can corrupt one process at a round, the total number of corrupted processes in any round is at most t. A corrupted process may behave arbitrarily, which means that even it is allowed not to follow the deployed algorithm. We also assume that, a corrupted process can recover and re-join the on-going execution after the corrupting agent left it. We refer the processes that are corrupted in the current round as *faulty*, and the processes that were faulty in the previous round, but no longer as *cured*. Also, we use the term *correct* to refer the processes that are neither faulty nor cured in the current round.

In the next subsection, we define two types of agents and their corresponding recovery models.

3.1.2 Agent Types and Recovery Models

Based on the severity of corruption, we can model two types of agents as follows:

- **Code-Erasable Free-moving (CEF) agent:** In addition to corruption, this agent do erase the local memory of the infected process including the code. It can also migrate from one process to another at any time during a round.
- **Code-Intact Free-moving (CIF) agent:** This agent can corrupt a process, but it can not erase its local memory. Also, it can migrate from one process to another at any time during a round.

We now construct two recovery models, one for each type of agent as follows: assume that an agent leaves a process p in round r.

- **CEF Agent Recovery (CEFAR) Model:** Process p recovers in round $r+1$. After recovery, it learns the code and the current state of computation from other processes by receiving messages that were sent in round $r+1$. Then, p starts participating in the on-going execution from round $r+2$.
- **CIF Agent Recovery (CIFAR) model:** Process p recovers to some predefined state in round $r+1$ and starts participating in the on-going execution in the same round. However, any process q that receives a message from p in this round can realize that p is a cured process and p can not contribute in a meaningful way to the on-going execution.

Note that, the CIFAR model allows the recovering processes to rejoin the on-going execution without learning the current state of computation while the CEFAR model requires that processes must learn it before reintegration.

3.1.3 Problem Definition

In mobile Byzantine consensus problem, each correct process p has an initial value v_p from the set $\mathcal{V}$ of all possible initial values, and decides a value v according to the following rules.

- **Termination:** Each correct process eventually irreversibly decides a value v.
- **Agreement:** The correct processes decide on the same value.
- **Unanimity:** If all correct processes have the same initial value v, then no correct process decides a value different from v.
- **Consistency maintenance:** Once agreement is reached among currently correct processes, it must be maintained among the (possibly different) correct processes.

Note that, any algorithm that solves the above problem is responsible not only for reaching the agreement, but for preserving the agreement among all correct processes forever. This is required, since even if agreement is reached at some point, the mobile agents can move to corrupt the correct processes and make the agreement disappear.

3.1.4 Notations

Let $\mathcal{V}$ be an ordered set of all possible proposal values. We introduce the default value $\perp$ such that $\perp \notin \mathcal{V}$ and $\perp < min(\mathcal{V})$. Let I be an vector in $(\mathcal{V} \cup \{\perp\})^n$. The number of occurrences of a value v in I is denoted by $\#_v(I)$.

3.2 Improved Mobile Byzantine Consensus Algorithm

3.2.1 Algorithm *MBC*

In this subsection, we present a mobile Byzantine consensus algorithm *MBC* for synchronous systems with *CEFAR* model. The algorithm is described in Fig. 3.1. It requires $n > 4t$ and at least one process remains uncorrupted for at least $3n$ rounds. This algorithm consists of phases, each phase is made of three rounds, namely, *Proposal round, Voting round, and Coordinator round* during which the processes exchange messages. Each message consists of

Algorithm MBC

1 : $val \leftarrow v$
2 : **Begin**
3 : **for** all s from 1 to ∞ **do**
4 : *coord-accept* $\leftarrow$ **True**
———— Proposal round of phase s ————
5 : $PV \leftarrow [\perp, \perp, ..., \perp]$
6 : Send (PROP, val) to all
7 : **for** all i **do**: **if** a message (PROP, v) is received from process i **then** $PV[i] \leftarrow v$
8 : **if** ($\exists v \neq \perp$: $\#_v(PV) \geq n - 2t$ and $\#_v(PV) + \#_\perp(PV) \geq n - t$) **then**
9 : $val \leftarrow v$
10 : **else**
11 : $val \leftarrow \perp$
12 : **end if**
———— Voting round of phase s ————
13 : $SV \leftarrow [\perp, \perp, ..., \perp]$
14 : Send (VOTE, val) to all
15 : **for** all i **do**: **if** a message (VOTE, v) is received from process i **then** $SV[i] \leftarrow v$
16 : **if** ($\exists v \neq \perp$: $\#_v(SV) > 2t$) **then**
17 : $val \leftarrow v$
18 : *coord-accept* $\leftarrow$ **False**
19 : **else if** ($\exists v \neq \perp$: $\#_v(SV) > t$) **then**
20 : $val \leftarrow v$
21 : **else**
22 : $val \leftarrow \perp$
23 : **end if**
———— Coordinator round of phase s ————
24 : $EV \leftarrow [\perp, \perp, ..., \perp][\perp, \perp, ..., \perp]$
25 : Send (ECHO, SV) to all
26 : **for** all i **do**: **if** a message (ECHO, sv) is received from process i **then** $EV[i] \leftarrow sv$
27 : **if** I am cured **then**
28 : Reconstruct()
29 : $coord \leftarrow s$ mod n;
30 : **if** ($\exists v \neq \perp$: $\#_v(EV[coord]) > t$) then $coord \leftarrow v$ else $coord \leftarrow \perp$ **end if**
31 : **if** *coord-accept* = **True then**
32 : $val \leftarrow \max\{coord\text{-}val, \min\{\mathcal{V}\}\}$
33 : **end for**
34 : **End**

Figure 3.1: Algorithm *MBC* for *CEFAR* model

```
Procedure Reconstruct()
35 :    Begin
36 :      for all i do
37 :        if ∃v ≠⊥ such that |{j|EV[j][i] = v}| > n − 2t then
38 :          SV[i] ← v
39 :        else
40 :          SV[i] ← ⊥
41 :        end if
42 :      end for
43 :      if (∃v ≠⊥: #_v(SV) > 2t) then
44 :        val ← v
45 :        coord-accept ← False
46 :      else if (∃v ≠⊥: #_v(SV) > t) then
47 :        val ← v
48 :      else
49 :        val ← ⊥
50 :      end if
51 :    End
```

Figure 3.2: Procedure *Reconstruct* for *CEFAR* model

values and a message tag (such as *PROP*, *VOTE* and *ECHO*) that indicates the name of the round in which it is sent. These messages can also include the algorithmic code and the current state of computation, etc., that help the recovering processes to correctly reintegrate into the on-going execution, but they are omitted for the sake of clarity. Remember that, by definition, in CEFAR model the cured processes do not send messages but they can receive messages. The three rounds are detailed as follows, where in the description, just 'process' means correct or cured one.

- *Proposal round:* The aim of this round is to end up in a situation where there is a single value v ($v \neq \perp$), such that every process adopts either v or $\perp$. To attain this goal, each currently correct process sends a message $< PROP, val >$ to provide the other processes with its value val and stores the received values in vector PV. Note that, since cured processes (if any) are silent, the entries that correspond to them in PV contain $\perp$. Each process adopts a value v if only if it appears at least $n-2t$ times in its vector PV and the sum of the number of occurrences

of v and $\perp$ in PV is at least $n - t$. Since any two sets of $n - t$ values will intersect at a correct process's value, it is ensured that there exists a single value v and all processes, that do not adopt $\perp$, will adopt the same value v.

- *Voting round:* In this round, each process checks for the existence of a unique value v. If such a value v exists, then it adopts v and chooses not to adopt the coordinator value in the next round, else it chooses to adopt the coordinator value in the following round. To do this, each correct process sends its value val in a message $< VOTE, val >$ to all processes and stores the received values in a vector SV. Then, if more than $2t$ of the votes are for some value v, then it adopts v and chooses not to accept the coordinator value by setting its variable *coord-accept* as "$False$". If more than t of the votes are for v, then it adopts v, otherwise, it adopts $\perp$. In these cases, processes choose to accept the coordinator value.

- *Coordinator round:* The aim of this round is to try to make all processes have the same value. In this round, each correct process sends a message $< ECHO, SV >$ to provide its vector SV to all processes, and stores the received vectors in EV. A cured process in this round executes the procedure *Reconstruct*, given in Fig. 3.2, to reconstruct the values for its variables val and *coord-accept*. Since, all processes are supposed to send their vectors, each process can compute the current coordinator value. Then, a process adopts the current coordinator value only if its variable *coord-accept* is "$True$". The secret of the algorithm is, when the coordinator is correct, at the end of the coordinator round, all processes have the same value v in its variable val regardless of whether they accept or ignore the coordinator value.

3.2.2 Correctness

In the following, for a given phase l, let $N_c^v(l)$ and $N_{cu}^v(l)$ respectively be the number of correct processes with $val = v$ and the number of cured processes that learnt v, at the end of the phase l.

Lemma 9 (Agreement) The correct processes decide on the same value.

Proof We say that the agreement is reached if all correct processes have the same value, say v, at the end of some phase l.
Claim. There exists a phase l such that at the end of l all correct and cured processes have the same value v. That is, the condition $N_c^v(l)+N_{cu}^v(l) \geq n-t$ holds. *End of the Claim.*
Once we establish this claim, the lemma directly follows.
Proof of the claim. Remember that, we have assumed one process, say k, that remains uncorrupted for at least $3n$ rounds. Let l be the phase in which this uncorrupted process k is the coordinator. Note that, as k is correct, it will send the same vector SV_k to all processes. There are two cases to consider (at line 31 of algorithm *MBC*):

- (Case 1:) *coord-accept* is $True$ for every process p (it means p chooses to adopt the coordinator value). Since all processes get the same vector SV_k from k, they compute the same value v at line 32 and adopt it. Hence, $N_c^v(l) + N_{cu}^v(l) \geq n - t$ holds. The claim follows.

- (Case 2:) *coord-accept* is $False$ for some process p (it means that p does not adopt the coordinator value, but other processes do adopt that value). Since *coord-accept* is $False$ at p, it follows that, there exists some value v such that $\#_v(SV_p) > 2t$, and hence p adopts v. Note that, in voting round, if two correct processes vote for $v \notin \perp$ and $v' \notin \perp$ respectively, then $v = v'$. Also, since there are at most t Byzantine processes, and cured processes do not cast vote, it is guaranteed that at coordinator k , $\#_v(SV_k) > t$ and $\#_{v'}(SV_k) \leq t$ holds for any value v' ($v' \notin \{v, \perp\}$). As a result, at every process the coordinator value is v. It follows that all processes (correct and cured) adopt the same value v regardless of whether they accept or ignore the coordinator value. The claim follows.

□

Lemma 10 (Consistency maintenance) Let $n \geq 4t + 1$. If at the end phase l, $N_c^v(l)+N_{cu}^v(l) \geq n-t$ holds for some value v, then $N_c^v(m)+N_{cu}^v(m) \geq n - t$ holds at all phases m ($m > l$).

Proof Assume that at the end of the phase l, $N_c^v(l) + N_{cu}^v(l) \geq n - t$ holds for some value v. In the proposal round of phase $l + 1$, at most t new faults can occur, and hence at least $n-2t$ correct processes will send v. As for each new fault there is a cured process, at each process p, $\#_v(PV_p) \geq n - 2t$ and $\#_v(PV_p) + \#_\perp(PV_p) \geq n - t$ holds. Hence, all correct and cured processes (that is, at least $n - t$ processes) set their values to v.

In the voting round, since at most t new faults can occur, at least $n-2t \geq 2t + 1$ processes vote for v. Hence, each p can collect at least $n - 2t$ votes for v (i.e., $\#_v(SV_p) > 2t$) . Also, as there are only t Byzantine faults, p can collect at most t votes for some value $v'(v' \notin \{v, \perp\})$. Therefore, p assigns v to val and "$False$" to *coord-accept*.

In the coordinator round, all (at least $n-2t$ correct) processes send their vector SV to all processes. Since the correct processes do not accept the coordinator value, their values remain as v. For each new fault, there is a cured process. When each cured process reconstructs its vector SV using procedure *Reconstruct*, it finds that $\#_v(SV) > 2t$. Thus, it adopts v and ignores the coordinator value. As a result, at the end of $l + 1$, $N_c^v(l + 1) + N_{cu}^v(l + 1) \geq n - t$ holds and the lemma follows. □

Lemma 11 (Unanimity) If all correct processes have the same initial value v, then no correct process decides a value different from v.

Proof Since at the beginning of first phase all correct processes (at least n-t) have the same initial value v, by using the same arguments as in lemma 10, we can show that at the end of first phase, the condition $N_c^v(1) + N_{cu}^v(1) \geq n - t$ holds and this situation continues in all the following phases. Hence, the lemma holds. □

Lemma 12 (Termination) Each correct process eventually irreversibly decides a value.

Proof From lemma 9, it is proved that all processes (correct and cured) decide on the same value v ($v \in \mathcal{V}$) at the end of the correct coordinator phase l. As well as, from lemma 10, it is proved that this decision value is never changed. Hence, each correct process eventually irreversibly decides a value. □

Theorem 4 Let $n > 4t$. The algorithm MBC described in Fig. 3.1 solves the mobile Byzantine consensus problem.

Proof The proof follows from lemmas 9, 10, 11, and 12. □

The algorithm *MBC* can work in the new model, *CIFAR model*, with a few lines of additional code given below:

```
6:    Send (PROP, val) to all
6.1:  if a message (MSG-TAG, *) received from t + 1 distinct processes then
6.2:    msg-tag ← MSG-TAG
6.3:    Learn s
6.4:  Case (msg-tag = PROP) : go to line 7
6.5:       (msg-tag = VOTE) : go to line 15
6.6:       (msg-tag = ECHO) : go to line 26
6.7:  End Case

52:   On Recovery
53:   val ←⊥
54:   go to line 2.
```

Theorem 5 Let $n > 4t$. The algorithm *MBC* described in 3.1 solves the mobile Byzantine consensus problem in synchronous systems with CIFAR model.

Proof Notice that, in CIFAR model, the cured processes participate in the on-going execution with some predefined state $\perp$. Hence, in every execution, the vectors computed by each process are exactly same to the corresponding ones resulted from the same execution in CEFAR model. Hence, lemmas 9, 10, 11 and 12 are exactly the same. Thus, the proof follows. □

3.3 Discussion and Open Problems

We conjecture that we can not improve the resilience more than $n > 4t$ in both *CEFAR* and *CIFAR* models, since a previous result [68] has proved that, even in a stronger model in which faults are static, $n > 4t$ is the necessary assumption to solve consensus with t benign failures and t Byzantine failures. Note that, benign failures are less severe than Byzantine failures. Examples for benign failures are omission failure and data-out-of-bound failure: when a process fails by omission, it forgets to send or receive messages, and when

it fails by data-out-of-bound, its proposal values are out of the set of all possible proposal values to consensus algorithms. In CEFAR model, we may consider the cured processes as omission failures since they do not send any messages. Similarly, in CIFAR model, the cured processes may be considered as data-out-of-bound failures since they send the value $\perp \notin \mathcal{V}$. Hence, we can say that our models also allow t benign and t Byzantine failures in every round. Above all, our models are weaker models in the sense that the set of faulty processes dynamically changes in every round.

This chapter leaves the following interesting questions:

- **Open Problem 1:** What is the lower bound on the number of processes for mobile Byzantine consensus in both *CEFAR* and *CIFAR* models?
- **Open Problem 2:** Can we improve the resilience further by reducing the speed of mobile agents? or We may ask : What is the relationship between resilience and the speed of agents?

Chapter 4

Summary and Future Directions

In this chapter, we summarize our contributions in this thesis and mention some problems for future investigations.

4.1 Summary of Contributions

We first summarize our main achievements concerning one-step Byzantine consensus.

- We have investigated one-step Byzantine consensus in asynchronous systems and have shown that the efficiency of the one-step Byzantine algorithms can be improved by adaptive condition-based approach. We have also shown that the impossibility of zero-degradation in one-step consensus can be circumvented by double-expedition property, a new concept introduced by us.

- We have proposed a novel generic adaptive doubly-expedited one-step Byzantine algorithm *DEX* that guarantees fast termination for a large number of inputs when there are fewer failures and provides both one- and two-step decisions without any additional stronger assumptions. Another important feature of our algorithm is that it is optimal in terms of the number of processes for one-step asynchronous consensus.

In addition, *DEX* can be instantiated with any condition-sequence pair satisfying the legality criteria defined by us. We also presented two examples of such legal condition-sequence pairs, namely *frequency-based pair* and *privileged-value-based pair*. One drawback of our algorithm is that it trades the decision scheme at third step for double-expedition property.

Next, we summarize our contribution for mobile Byzantine Consensus.

- We have improved the result of Garay [43] without any additional assumption. That is, we have proposed a mobile Byzantine consensus algorithm *MBC* for synchronous systems prone to malicious agents that can corrupt processes and move at full speed from processes to processes to corrupt new processes before the currently corrupted processes recover. This algorithm improves on the bound t, namely it requires $n > 4t$, in stead of $n > 6t$ (Garay's result). In addition, we show that our algorithm can also work for a new model in which any process can restart the algorithm immediately but with some predefined state after its recovery.

4.2 Future Directions

- **Mobile Byzantine consensus with Homonyms:** In this thesis, we have assumed a network in which processes have distinct identities. However, it might be a very strong assumption in systems like [62, 67], due to the fact that authenticated unique identifiers are typically achieved through collision free hash functions and these schemes may be vulnerable to malicious attacks, since inputs that produce the same output for some such functions can be found efficiently [72]. In addition, in many cases, processes (users) may wish to preserve their privacy by remaining anonymous. However, in a fully anonymous system where processes have no identifiers at all Byzantine consensus is simply impossible. By using a limited number of identifiers, that is by assigning l identifiers to n processes where $l \leq n$, one can preserve some level of anonymity and hide, to some extent, the association between

users and identifiers. In the literature, the processes with the same identifier are called as *Homonyms* [26]. There have been very few works [26] that attempt to design protocols that work with homonyms. We may study the mobile Byzantine consensus in networks with homonyms.

- **Incomplete Network:** In this thesis, we restricted our network to complete network. But in real life it is quite impractical to have an underlying network as complete network where every two processes can communicate between them through their private reliable channel. There have been very little attempt [1, 28, 29, 52] to characterize and find feasible solutions for Byzantine consensus over incomplete networks. Thus it is very important to pursue further research in that network model.

- **Mobile Network:** While the topology of the network is assumed to remain unchanged throughout the runtime of the protocol in this thesis, such an assumption may not always hold in practice. We infer that the ideas to deal with mobile adversaries over static networks can be used to thwart static adversaries in mobile networks (over the same set of processes) - this is because, the disappearing of an edge can be treated as that edge being newly fail-stop corrupted, while the reappearing of a new edge may be treated as the curing of that infected edge.

- **Unknown Network:** In this thesis, we assumed that the set of processes that participate in consensus execution is static and known. But in a scenario of unknown networks, the set and number of participants are previously unknown. Each participant knows only its neighbors. There have been very few works that study consensus in the context of dynamic and self-organizing systems characterized by unknown network [1, 20, 44]. We may further study Byzantine consensus in unknown networks.

Acknowledgements

This thesis is the end of my journey in obtaining my Ph.D. At this moment of accomplishment, it is my pleasure to express my thanks to all those who contributed in many ways to the success of this study and made it an unforgettable experience for me.

First of all, I would like to express my deep and sincere gratitude to my wonderful supervisor Prof. Koichi Wada for accepting me in his laboratory and for his support in dissecting distributed consensus. This work would not have been possible without his guidance, support and encouragement. His perpetual energy, wide knowledge and enthusiasm in research have been of great value for me. In addition, he was always accessible and willing to help his students. As a result, research life became smooth and rewarding for me.

I am also extremely indebted to my co-supervisor Prof. Taisuke Izumi who sharpened my thinking, refined my arguments, improved my writing and, most of all, provided both support and kindness at all times. He was and remains my best role model for a scientist, mentor, and teacher.

I take this opportunity to sincerely acknowledge the Japanese Government for providing financial assistance in the form of MONBUKAGAKUSHO (MEXT) scholarship which buttressed me to perform my work comfortably. I also acknowledge Nagoya Institute of Technology for providing me an excellent research atmosphere. I would like to thank Hori foundation for providing financial support for my travel to an international workshop.

I will forever be thankful to my former college research advisor Prof. D.I. George Amalarethinam. He was the reason why I decided to pursue my doctoral course in Japan. I will never forget the philosophical conversations that we had many times. As I look back now, I see how those conversations boosted my spirits, changed many aspects of my life immensely.

I gratefully acknowledge my lab mate Dr. Samia Souissi for her understanding, encouragement and personal attention which have provided good and smooth basis for my Ph.D. tenure. She has a tremendous influence on my professional as well as personal development. I owe my loving thanks to my close friends Khairunnisa, Fathima, Noorjahan, Vahida and Hafeeza who stood by me through the ups and downs of my life. I will never forget about those endless gossips, laughs that we had. Their friendship is truly beyond words of acknowledgement.

My sincere thanks to my in-laws (N. Sirajudeen and Late S. Rahmathunisa) and sister-in-law (Dr. S. Shameema Banu) for their unconditional support. They have always encouraged me for my study and above all they have unconditionally supported me in all that I wanted to do in my life.

Now I would like to thank five remarkable people who had, are having and will continue to have a tremendous influence on my life: my parents A. Mohamed Hanifa and M. Mallika Jan , my husband S. Mohamed Farook, my kids M. Thasleem Arifa and M. Mohamed Irfan. They have lost a lot due to my research abroad. Without their encouragement and understanding it would have been impossible for me to finish this work.

Above all, I would like to thank God, the almighty, for having made everything possible by giving me strength and courage to do this work.

Bibliography

[1] E. A. P. Alchieri, A. N. Bessani, J. S. Fraga, and F. Greve. Byzantine consensus with unknown participants. In *Proc. of the 12th International Conference on Principles of Distributed Systems*, pages 22–40, 2008.

[2] B. Altmann, M. Fitzi, and U. M. Maurer. Byzantine agreement secure against general adversaries in the dual failure model. volume 1693 of LNCS, pages 123–137, 1999.

[3] H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics.* Wiley, 2004.

[4] M. Ben-Or. Another advantage of free choice: Completely asynchronous agreement protocols. In *Proc. of the Second Annual ACM SIGACT-SIGOPS Symposium on Princiles of Distributed Computing*, pages 27–30, 1983.

[5] M. Ben-Or, S. Goldwasser, and A. Wigderson. Completeness theorems for non-cryptographic fault-tolerant distributed computation(extended abstract). In *Proc. of the 20th Annual ACM Symposium on Theory of Computing*, pages 1–10, 1988.

[6] M. Ben-Or, E. Pavlov, and V. Vaikuntanathan. Byzantine agreement in the full-information model in o(log n) rounds. In *Symposium on Theory of Computing*, pages 179–186, 2006).

[7] P. Berman and J. A. Garay. Asymptotically optimal distributed consensus (extended abstract). In *Proc. of the 16th International Colloquium on Automata, Languages and Programming*, pages 80–94, 1989.

[8] P. Berman, J. A. Garay, and K. J. Perry. Towards optimal distributed consensus (extended abstract). In *Proc. of 30th Annual Symposium on Foundations of Computer Science*, pages 410 –415, 1989.

[9] A. Bessani, P. Sousa, M. Correia, N. Neves, and P. Verissimo. The crutial way of critical infrastructure protection. *IEEE Security and Privacy*, 6(6):44–51, 2008.

[10] M. Biely and M. Hutle. Consensus when all processes may be byzantine for some time. *Journal of Theoretical Computer Science*, 412(33):4260–4272, 2011.

[11] G. Bracha. An asynchronous $\lfloor (n-1)/3 \rfloor$ -resilient consensus protocol. In *Proc. of the Third Annual ACM Symposium on Princiles of Distributed Computing*, pages 154–162, 1984.

[12] G. Bracha. Asynchronous byzantine agreement protocols. *Information and Computation*, 75(2):130–143, 1987.

[13] G. Bracha and S. Toueg. Asynchronous consensus and broadcast protocols. *Journal of ACM*, 32(4):824–840, 1985.

[14] V. Brasileiro, F. Greve, A. Mostéfaoui, and M.Raynal. Consensus in one communication step. In *Proc. of the 6th International Conference on Parallel Computing Technologies*, volume 2127 of LNCS, pages 42–50, 2001.

[15] H. Buhrman, J. A. Garay, and J. Hoepman. Optimal resiliency against mobile faults. In *Proc. 25th International Symposium on Fault-Tolerant Computing (FTCS'95)*, pages 83–89, 1995.

[16] C. Cachin, K. Kursawe, F. Petzold, and V. Shoup. Secure and efficient asynchronous broadcast protocols (extended abstract). In *Kilian, J., editor, Advances in Cryptology: CRYPTO 2001*, volume 2139 of LNCS, pages 524–541, 2001.

[17] R. Canetti and T. Rabin. Fast asynchronous byzantine agreement with optimal resilience. In *Proc. of the Twenty-Fifth Annual ACM Symposium on Theory of Computing*, pages 42–51, 1993.

[18] M. Castro and B. Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems*, 20(4):398–461, 2002.

[19] M. Castro and B. Liskov. Practical byzantine fault tolerance and proactive recovery. *ACM Transactions on Computer Systems (TOCS)*, 20(4):398–461, 2002.

[20] D. Cavin, Y. Sasson, and A. Schiper. Consensus with unknown participants or fundamental self-organization. In *Proc. of the 3rd Interntional Conference on AD–HOC Networks & Wireless (ADHOC-NOW'04)*, pages 135–148, 2004.

[21] T. Chandra and S. Toueg. Unreliable failure detectors for reliable distributed systems. *Journal of the ACM*, 43(2):225–267, 1996.

[22] B. Chun, P. Maniatis, S. Shenker, and J. Kubiatowicz. Attested append-only memory: making adversaries stick to their word. In *Proc. of the 21st ACM Symposium on Operating Systems Principles*, pages 189–204, 2007.

[23] J. Considine, M. Fitzi, M. K. Franklin, L. A. Levin, U. M. Maurer, and D. Metcalf. Byzantine agreement given partial broadcast. *Journal of Cryptology*, 18(3):191–217, 2005.

[24] M. Correia, N. Neves, and P. Verissimo. From consensus to atomic broadcast: Time-free byzantine-resistant protocols without signatures. *Computer Journal*, 41(1):82–96, 2006.

[25] M. Correia, N. F. Neves, L. C. Lung, and P. Verissimo. Low complexity byzantine resilient consensus. *Distributed Computing*, 17(3):237–249, 2005.

[26] C. Delporte-Gallet, H. Fauconnier, R. Guerraoui, A. Kermarrec, E. Ruppert, and H. Tran-The. Byzantine agreement with homonyms. In *Principles of Distributed Computing*, pages 21–30, 2011.

[27] D. Dobre and N. Suri. One-step consensus with zero-degradation. In *Proc. of the International Conference on Dependable Systems and Networks(DSN'06)*, pages 137–146, 2006.

[28] D. Dolev. Unanimity in an unknown and unreliable environment. In *Proc. 22nd IEEE Symposium on Foundations of Computer Science*, pages 159–168, 1981.

[29] D. Dolev. The byzantine generals strike again. *Journal of Algorithms*, 3(1):14–30, 1982.

[30] D. Dolev, C. Dwork, and L. Stockmeyer. On the minimal synchronism needed for distributed consensus. *Journal of the ACM*, 34(1):7797, 1987.

[31] D. Dolev, M. J. Fischer, R. J. Fowler, N. A. Lynch, and H. R. Strong. An efficient algorithm for byzantine agreement without authentication. *Information and Control*, 52(3):257–274, 1982.

[32] D. Dolev and R. Reischuk. Bounds on information exchange for byzantine agreement. *Journal of ACM*, 32(1):191–204, 1985.

[33] D. Dolev, R. Reischuk, and H. R. Strong. Early stopping in byzantine agreement. *Journal of ACM*, 37(4):720–741, 1990.

[34] D. Dolev and H. R. Strong. Authenticated algorithms for byzantine agreement. *SIAM Journal on Computing*, 12(4):656–666, 1983.

[35] P. Dutta and R. Guerraoui. Fast indulgent consensus with zero degradation. In *Proc. of the 4th European Dependable Computing Conference on Dependable Computing*, volume 2485 of LNCS, pages 191–208, 2002.

[36] C. Dwork, N. Lynch, and L. Stockmeyer. Consensus in the presence of partial synchrony. *Journal of the ACM*, 35(2):288–323, 1988.

[37] P. Feldman and S. Micali. Byzantine agreement in constant expected time (and trusting no one). In *Proc. of 26th Annual Symposium on Foundations of Computer Science*, pages 267–276, 1985.

[38] P. Feldman and S. Micali. An optimal protocol for synchronous byzantine agreement. *SIAM Journal of Computing*, 26:873–933, 1997.

[39] M. Fischer, N. Lynch, and M. Paterson. Impossibility of distributed consensus with one faulty process. *Journal of ACM*, 32(2):374–382, 1985.

[40] M. Fitzi, D. Gottesman, M. Hirt, T. Holenstein, and A. Smith. Detectable byzantine agreement secure against faulty majorities. In *Proc. of the Twenty-First Annual ACM Symposium on Principles of Distributed Computing*, pages 118–126, 2002.

[41] M. Fitzi and U. Maurer. Efficient byzantine agreement secure against general adversaries. In *Proc. of Distributed Computing (DISC '98)*, volume 1499 of LNCS, 1998.

[42] R. Friedman, A. Mostefaoui, and M. Raynal. Simple and efficient oracle-based consensus protocols for asynchronous byzantine systems. *IEEE Transactions on Dependable and Secure Computing*, 2(1):46–56, 2005.

[43] J. Garay. Reaching (and maintaining) agreement in the presence of mobile faults. In *Proc. 8th International Workshop on Distributed Algorithms*, volume 857 of LNCS, pages 253–264, 1994.

[44] F. Greve and S. Tixeuil. Knowledge connectivity vs. synchrony requirements for fault-tolerant agreement in unknown networks. In *Proc. of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks(DSN '07)*, pages 82–91, 2007.

[45] R. Guerraoui and M. Raynal. The information structure of indulgent consensus. *IEEE Transactions on Computers*, 53(4):453–466, 2004.

[46] R. Guerraoui and A. Schiper. The generic consensus service. *IEEE Transactions on Software Engineering*, 27(1):29–41, 2001.

[47] V. Hadzilacos and S. Toueg. A modular approach to fault-tolerant broadcasts and related problems. Technical report, 1994.

[48] T. Izumi and T. Masuzawa. Condition adaptation in synchronous consensus. *IEEE Transactions on Computers*, 55:843–853, 2006.

[49] T. Izumi and T. Masuzawa. One-step consensus solvability. In *Proc. of the 22nd international symposium on Distributed Computing(DISC'06)*, volume 4167 of LNCS, pages 224–237. Springer, 2006.

[50] I. Keider and S. Rajsbaum. On the cost of fault-tolerant consensus when there are no faults. *SIGACT News*, 32(2):45–63.

[51] L. Lamport. The weak byzantine generals problem. *Journal of ACM*, 30(3):668 –676, 1983.

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

[53] N. A. Lynch. *Distributed Algorithms*. Morgan Kaufmann, 1996.

[54] D. Malkhi and M. Reiter. Unreliable intrusion detection in distributed computations. In *Proc. of the 10th Computer Security Foundations Workshop*, pages 116–124, 1997.

[55] A. Mostefaoui, S. Rajsbaum, and M. Raynal. Conditions on input vectors for consensus solvability in asynchronous distributed systems. *Journal of the ACM*, 50(6):922–954, 2003.

[56] A. Mostéfaoui, S. Rajsbaum, and M. Raynal. Using conditions to expedite consensus in synchronous distributed systems. In *Proc. of the 17th international symposium on Distributed Computing(DISC'03)*, volume 2848 of LNCS, pages 249–263, 2003.

[57] A. Mostefaoui, M. Raynal, C. Travers, S. Pattersona, D. Agrawal, and A. Abbadi. From static distributed systems to dynamic systems. In *Proc. of the 24th IEEE Symposium on Reliable Distributed Systems*, pages 109–118, 2005.

[58] N. F. Neves, M. Correia, and P. Verissimo. Solving vector consensus with a wormhole. *IEEE Transactions on Parallel and Distributed Systems*, 16(12):1120–1131, 2005.

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

[60] M. O. Rabin. Randomized byzantine generals. In *Proc. of the 24th Annual IEEE Symposium on Foundations of Computer Science*, pages 403–409, 1983.

[61] D. V. S. Ravikant, M. Venkitasubramaniam, V. Srikanth, K. Srinathan, and C. P. Rangan. On byzantine agreement over (2,3)-uniform hypergraphs. In *Proc. of the 18th international symposium on Distributed Computing(DISC'04)*, volume 3274 of LNCS, pages 450–464, 2004.

[62] A. Rowstron and P. Druschel. Pastry: Scalable, decentralized object location and routing for large-scale peer-to-peer systems. In *Proc. of Middleware'01 : IFIP/ACM International Conference on Distributed Systems Platforms*, volume 2218 of LNCS, pages 329–350, 2001.

[63] N. Santoro and P. Widmayer. Time is not a healer. In *Proc. 6th Annual Symposium on Theor. Aspects of Computer Science(STACS89)*, volume 349 of LNCS.

[64] U. Schmid, B. Weiss, and I. Keidar. Impossibility results and lower bounds for consensus under link failures. *SIAM Journal on Computing*, 38:1912–1951, 2009.

[65] F. Schneider. Implementing faul-tolerant services using the state machine approach: A tutorial. *ACM Computing Surveys*, 22(4):299–319, 1990.

[66] Y. Song and R. Renesse. Bosco: One-step byzantine asynchronous consensus. In *Proc. of the 22nd international symposium on Distributed Computing(DISC'08)*, volume 5218 of LNCS.

[67] I. Stoica, R. Morris, D. Karger, M. K. Frans, and H. Balakrishnan. Chord: A scalable peer-to-peer lookup service for internet applications. *SIGCOMM Computer Commununication Review*, 31:149–160, 2001.

[68] P. Thambidurai and P. You-Keun. Interactive consistency with multiple failure modes. In *Proc. Reliable Distributed Systems*, pages 93 – 100, 1988.

[69] S. Toueg, K. J. Perry, and T. K. Srikanth. Fast distributed agreement. *SIAM Journal of Computing*, 16(3):445–457, 1987.

[70] R. Turpin and B. A. Coan. Extending binary byzantine agreement to multivalued byzantine agreement. *Information Processing Letters*, 18(2):73–76, 1984.

[71] G. Veronese, M. Correia, A. Bessani, and L. Lung. Highly-resilient services for critical infrastructures. In *Proc. of the Embedded Systems and Communications Security Workshop*, 2009.

[72] X. Wang and H. Yu. How to break md5 and other hash functions. In *EUROCRYPT, volume 3494 of LNCS*, pages 19–35, 2005.

[73] J. Yin, J. Martin, A. Venkataramani, L. Alvisi, and M. Dahlin. Separating agreement from execution for byzantine fault-tolerant services. In *Proc. of the 19th ACM Symposium on Operating Systems Principles*, pages 253–267, 2003.