FAULT-TOLERANT CONSENSUS IN DIRECTED GRAPHS AND CONVEX HULL CONSENSUS

BY

LEWIS TSENG

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Computer Science
in the Graduate College of the
University of Illinois at Urbana-Champaign, 2016

Urbana, Illinois

Doctoral Committee:

Professor Nitin H. Vaidya, Chair
Professor Chandra Chekuri
Associate Professor Indranil Gupta
Professor Jennifer L. Welch, Texas A&M University

Abstract

As distributed systems nowadays scale to thousands or more of nodes, fault-tolerance becomes one of the most important topics. This dissertation studies the fault-tolerance aspect of the *consensus* algorithm, which is a fundamental building block for the distributed systems. Particularly, the dissertation has the following two main contributions on fault-tolerant consensus in message-passing networks:

- We explore various fault-tolerant consensus problems under different fault models in communication networks that are modeled as *arbitrary directed* graphs, i.e., two pairs of nodes may not share a bi-directional communication channel, and not every pair of nodes may be able to communicate with each other directly or indirectly. We prove the *tight* condition of the underlying communication graphs for solving each of the consensus problem, i.e., the necessary condition is equal to the sufficient condition.
- We propose a new consensus problem – convex hull consensus – in which the input is a vector of reals in the d-dimensional space, and the output is a convex polytope contained within the convex hull of all inputs at fault-free nodes. For asynchronous systems, we present an approximate convex hull consensus algorithm with optimal fault tolerance that reaches consensus on optimal output polytope under crash fault model. Convex hull consensus may be used to solve related problems, such as vector consensus and function optimization with the initial convex hull as the domain.

To Father, Mother and Wife.

Acknowledgments

I have spent my past ten and half years in Champaign. “Wow” is the most reaction I got from people. However, I have quite enjoyed my time here, because I have met lots of good people and friends. I would like to use this space to thank all of them. Specifically, I want to thank Tyrone L. Roach (a.k.a. Dr. Roach), who played basketball with me and helped me assimilate into American culture.

My deepest gratitude is to my advisor, Prof. Nitin H. Vaidya, for his guidance and support. I appreciate all his contributions of time, ideas and funds which made this dissertation possible. Beyond research, he taught me how to think critically, how to express ideas, and most importantly, how to be an ethical researcher – these are among the most important things that I have learned during my PhD journey.

I also thank Prof. Chekuri, Prof. Gupta and Prof. Welch for serving on my doctoral committee, and providing their insightful comments. Particularly, Prof. Chekuri has given me many useful advices since I was a dumb undergraduate student. (Hopefully, I am a bit smarter now.) I also enjoyed my time working as a teaching assistant for Prof. Chekuri.

My research was financially supported by the National Science Foundation, U.S. Army Research Office and the Boeing Company. My colleagues in the Distributed Algorithms and Wireless Networking Group have been very helpful in many ways; I thank all of them. I also thank Carol Wisniewski for her kind help with various administrative matters.

Most importantly, I would like to thank my wife Yutien (Casey) Cheng for her support and love. Without her constant reminder, my dissertation may not finish in time! I also thank my parents, Yiping Tseng and Hueichu Liao, for their faith in me. Finally, I thank my dearest friends, who I would rather call my brothers – Hao (Joseph) Chen and Yung-Hsin Chang, for making me a better man. Joseph, may your soul rest in peace.

Table of Contents

Chapter 1

Introduction

In recent years, we have seen tremendous growth in large-scale distributed systems, such as networked multi-agent systems [56, 38], distributed robots [63, 59] and distributed storage systems [21, 22, 25, 1, 44, 16]. These systems are usually designed to scale to thousands or more of nodes.[1] As a result, failures become a norm rather than an exception [25, 23], and it is important to design systems that *tolerate failures*. In other words, the systems should behave correctly even if some failures happen. *Consensus* algorithm is one of the fundamental building blocks for fault-tolerant distributed systems. Roughly speaking, fault-tolerant *consensus* allows multiple nodes to coordinate with each other in the presence of faults, and it has received significant attention over the past three decades [5, 52] since the seminal work by Pease, Shostak, and Lamport [60]. Fault-tolerant consensus has been studied extensively in academia, e.g., [29, 31, 2, 45], and has been adopted in many practical systems for various usages, such as state-machine replication [68], lock service [15], data aggregation [40], decentralized estimation [66], and flocking [38]. This dissertation studies fault-tolerant consensus in point-to-point message-passing network, and has two main contributions: fault-tolerant consensus in arbitrary directed graphs (Chapters 3, 4, 5, 6, 7, 8), and a new type of consensus problem – convex hull consensus (Chapter 9). In the rest of this Chapter, we briefly discuss the main contributions, formally define the system and fault models, and present the dissertation outline.

1.1 Main Contributions

1.1.1 Fault-tolerant Consensus in Arbitrary Directed Graphs

We consider the consensus problem in a point-to-point message-passing network, which is modeled as a *directed* graph. Thus, we will often use the terms *graph* and *network* interchangeably. Consensus problem has been studied extensively both in complete graphs [60, 29, 31, 2, 45] and in undirected graphs [32, 27]. However, the conditions identified in these papers are not adequate to fully characterize the *directed* graphs

[1] Throughout the discussion of this dissertation, we will use the term “node” to represent a computation entity, such as servers, devices or processes.

in which consensus is possible. In arbitrary directed communication network, not every pair of nodes shares a communication channel, and the communication channel between neighboring nodes are not necessarily bi-directional. This research topic is motivated by the presence of directed links in wireless networks. However, we believe that the results are of independent interests as well. This dissertation explores various consensus problems in arbitrary directed graphs and proves the tight condition of the underlying communication graphs in which each consensus problem is solvable. Particularly, for each consensus problem, we derive a necessary condition that must be satisfied by the communication graph so that a correct consensus algorithm exists. For graphs that satisfy this necessary condition, we provide a correct consensus algorithm, proving that the necessary conditions are also sufficient (i.e., tight condition). As we will see later, the algorithms for directed graphs presented in this dissertation are substantially different from those previously developed for undirected graphs, e.g., [60, 29, 31, 2, 45, 32, 27].

In the fault-tolerant consensus problem [5, 52], each node is given an *input*, and after a finite amount of time, each fault-free node should produce an *output*, which satisfies appropriate *validity* and *agreement* conditions. We limit our consideration to *scalar* input and output in this part of the dissertation. There are four dimensions to formally define a consensus problem – output type, fault model, system synchrony, and algorithm type. We briefly discuss each of the dimensions below. The formal definitions will be presented in each relevant Chapter.

Output Type We consider both *exact* consensus [60, 45] and *approximate* consensus [29, 31]. Intuitively, exact consensus requires the fault-free nodes to agree on exactly the *same* output; whereas, approximate consensus requires the fault-free nodes to produce outputs within a certain constant ϵ $(\epsilon > 0)$ of each other. Obviously, output type is related to how agreement and validity conditions are defined. The formal definitions of the agreement and validity conditions are discussed later in each relevant Chapter.

Fault Model We explore both node and link faults.

- For node faults, we address crash faults and Byzantine faults both. A node that suffers crash failure simply stops taking steps; whereas, a Byzantine faulty node may misbehave arbitrarily [5, 52, 60]. For the location of node failures, we consider f-total fault model (up to f nodes in the system may be faulty), f-local fault model (up to f incoming nodes of a fault-free node may be faulty) and generalized fault model (a fault model specifying potential locations of failures).

- For link faults, we consider transient Byzantine link failures [64, 65].

The formal definitions of the fault models are presented later in Chapter 1.2.

System Synchrony We study both synchronous and asynchronous systems.

- In synchronous systems, nodes proceed in a lock-step fashion, and we consider both exact and approximate consensus. The notion of approximate consensus is of interest in *synchronous* systems, since approximate consensus can be achieved using distributed algorithms that do not require complete knowledge of the network topology [9, 29].
- In asynchronous systems, no known bound on the communication delay or processing speed exists. Moreover, it is known that exact consensus is impossible to achieve in asynchronous systems [33]; hence, we consider only approximate consensus.

Algorithm Type We study two types of consensus algorithms: (i) *general algorithms*: nodes have complete knowledge of the network topology, and (ii) *iterative algorithms*: nodes are constrained to only have knowledge of the *local* network topology, particularly their immediate neighbors. General algorithms usually achieve consensus more efficiently; however, they require extra information and overhead, such as topology knowledge and routing mechanism. Therefore, we also consider iterative algorithms. Moreover, as we will see later, each node only needs to maintain minimal state in each iteration.

Summary of Consensus Problems For each consensus problem, our goal is to characterize the *tight* necessary and sufficient conditions on the underlying communication network for solving the consensus. Table 1.1 summarizes the results on fault-tolerant consensus under f-total fault model. Recall that in f-total fault model, up to f nodes may become faulty in the system. For problems that are solved previously, the corresponding entries cite the relevant papers. For problems solved in this dissertation, the entries list the dissertation chapters that discuss the tight condition. The table also identifies two problems that remain open. Below, we also list all the consensus problems that we explore in this first part of the dissertation. Note that the consensus problems of the first five bullets are included in Table 1.1 as well.

- *General algorithms* for f-total fault model:

  1. Exact crash-tolerant consensus in synchronous systems (Chapter 3.4)
  2. Approximate crash-tolerant consensus in asynchronous systems (Chapter 3.5)
  3. Exact Byzantine consensus in synchronous systems (Chapter 3.6)

- *Iterative algorithms*:

  1. Approximate Byzantine consensus in both synchronous and asynchronous systems under f-total fault model (Chapter 4)

Table 1.1: Results on fault-tolerant consensus under f-total fault model

<table>
<tr><th>Fault Model</th><th>System</th><th>Output</th><th>Graph</th><th>General Alg.</th><th>Iterative Alg.</th></tr>
<tr><td rowspan="6">Crash</td><td rowspan="4">Sync.</td><td rowspan="2">Exact</td><td>Undirected</td><td colspan="2">[5, 52]</td></tr>
<tr><td>Directed</td><td colspan="2">Chapter 3.4</td></tr>
<tr><td rowspan="2">Approx.</td><td>Undirected</td><td colspan="2">[9, 38]</td></tr>
<tr><td>Directed</td><td colspan="2">[9, 38, 17, 18]</td></tr>
<tr><td rowspan="2">Async.</td><td rowspan="2">Approx.</td><td>Undirected</td><td colspan="2">[5, 52]</td></tr>
<tr><td>Directed</td><td>Chapter 3.5</td><td>Chapter 7</td></tr>
<tr><td rowspan="6">Byzantine</td><td rowspan="4">Sync.</td><td rowspan="2">Exact</td><td>Undirected</td><td colspan="2">[60, 32]</td></tr>
<tr><td>Directed</td><td>Chapter 3.6</td><td><i>Open</i></td></tr>
<tr><td rowspan="2">Approx.</td><td>Undirected</td><td colspan="2">[32, 27, 29]</td></tr>
<tr><td>Directed</td><td>[72]</td><td>Chapter 4</td></tr>
<tr><td rowspan="2">Async.</td><td rowspan="2">Approx.</td><td>Undirected</td><td colspan="2">[32, 27, 29]</td></tr>
<tr><td>Directed</td><td><i>Open</i></td><td>Chapter 4</td></tr>
</table>

2. Approximate crash-tolerant consensus in asynchronous systems under f-total fault model (Chapter 7)

3. Approximate Byzantine consensus in synchronous systems under generalized fault model (Chapter 5)

4. Approximate consensus in synchronous systems under transient Byzantine link fault model (Chapter 6)

Inspired by famous matrix tools [38, 93, 9, 34], we develop a proof technique based on the notion of transition matrix, and we use the technique to prove the correctness of fault-tolerant iterative algorithms. The correctness proofs of these algorithms share some similarities because of the nature of the iterative algorithm. We will point out these similarities when we present the proofs. These iterative algorithms share the same iterative structures; however, the implementation details of each algorithm are quite different from each other to accommodate different types of faulty behavior.

One closely related problem is *reliable broadcast*, in which a single fault-free source needs to transmit an input to all the other fault-free nodes (also called peers). For the broadcast problem, we consider f-local model, where up to f incoming neighbors of each fault-free node may become Byzantine faulty. This dissertation also proves the tight condition for the following problem:

- Reliable broadcast using certified propagation algorithm in the presence of Byzantine peers (Chapter 8)

1.1.2 New Consensus Problem: Convex Hull Consensus

The traditional fault-tolerant consensus problem formulation assumes that each node has a scalar input. As a generalization of this problem, recent work [54, 88, 87] has addressed *vector* consensus (also called multidimensional consensus) in the presence of Byzantine faults, wherein each node has a d-dimensional vector of reals as input ($d \geq 1$). Define the *initial convex hull* as the convex hull of the inputs at fault-free nodes. Vector consensus requires the nodes to reach consensus on a d-dimensional vector within the initial convex hull. We propose the problem of *convex hull consensus*, where the output at each node is a *convex polytope* contained within the initial convex hull. Intuitively, the goal is to reach consensus on the "largest possible" polytope, allowing the node to estimate the domain of inputs at the fault-free nodes. For the convex hull consensus problem, we consider complete graphs with reliable links, in which every pair of nodes can communicate with each other reliably. In the asynchronous systems under crash fault models, we present an approximate convex hull consensus algorithm. The algorithm has optimal fault tolerance and reaches consensus on largest possible output polytope. We can also use the simulation techniques [20, 5] to transform our algorithm so that it tolerate Byzantine faults. Convex hull consensus may be used to solve related problems, such as vector consensus [54, 88] and function optimization with the initial convex hull as the domain. We present the results on convex hull consensus in Chapter 9.

1.2 Models

We formally introduce the models used in this dissertation. The models are typical in the literature [5, 52].

1.2.1 System Model

We consider a point-to-point message-passing network in which nodes are connected by *directed* links. The communication network is *static*, and it is represented by a simple directed graph $G(\mathcal{V}, \mathcal{E})$, where $\mathcal{V}$ is the set of n nodes, and $\mathcal{E}$ is the set of directed edges between the nodes in $\mathcal{V}$. We assume that $n \geq 2$, since the consensus problem for $n = 1$ is trivial. Node i can transmit messages to another node j if directed edge (i, j) is in $\mathcal{E}$. Each node can send messages to itself as well; however, for convenience, we exclude self-loops from set $\mathcal{E}$. Note that we consider arbitrary directed graphs; thus, not every pair of nodes may be connected. In other words, the communication network may be incomplete.

All the communication links are assumed to be reliable, FIFO (first-in first-out) and deliver each transmitted message exactly once – the only exception is the link fault model considered in Chapter 6, where a subset of links may be faulty, and faulty links may not transmit messages reliably. When node i wants to send message M on link (i, j) to node j, it puts the message M in a send buffer for link (i, j). No further operations are needed at node i; the mechanisms for implementing reliable, FIFO and exactly-once semantics are transparent to the nodes. When a message is delivered on link (i, j), it becomes available to node j in a receive buffer for link (i, j). We will often use the terms *edge* and *link* interchangeably.

System Synchrony We consider both synchronous and asynchronous systems [5, 52]. In synchronous systems, there is a known finite bound on both nodes' processing operations and on communication delay over communication channel. In other words, nodes proceed in a lock-step fashion in synchronous systems. In contrast, there is no such bound in asynchronous systems. Thus, a very slow node cannot be distinguished from a faulty node in asynchronous systems – this is the main issue why it is impossible to reach exact consensus in asynchronous systems with node faults [32]. Hence, we consider only approximate consensus in asynchronous systems.

1.2.2 Fault Model

This dissertation explores several types of fault models, including link and node faults. All chapters consider node faults with the exception of Chapter 6. For node faults, there are two dimensions under consideration: location and behavior of the failures.

- *Location*: We consider f-total fault model (up to f nodes in the system may be faulty), f-local fault model (up to f incoming neighbors of each fault-free node may be faulty) and generalized fault model (location of the failures is specified by a fault location model as discussed in Chapter 5). Note that both f-total and f-local fault models are special cases of the generalized fault model.

- *Behavior*: We consider both crash and Byzantine faults. Crash fault is considered to be a benign fault model, because a crashed node simply stops executing the algorithm without sending any tampered messages. In contrast, a Byzantine faulty node may misbehave arbitrarily. Possible misbehavior includes sending incorrect and mismatching (or inconsistent) messages to different neighbors. The Byzantine nodes may potentially collaborate with each other. Moreover, the Byzantine nodes are assumed to have a complete knowledge of the execution of the algorithm, including the states of all the nodes, contents of messages the other nodes send to each other, the algorithm specification, and the network topology.

For link failures, we adopt transient Byzantine link fault [64, 65] models, in which an omniscient adversary controls up to f *directed* communication links at any point of time, but the nodes are assumed to be *fault-free*. The faulty link may misbehave arbitrarily. Moreover, a different set of links can be faulty at different time. The detail of the fault model is presented in Chapter 6. For brevity, "crash fault" and "Byzantine fault" refer to node faults only. We will explicitly use "link fault" in the case when links may misbehave.

1.3 Dissertation Outline

- In Chapter 2, we present the related work.
- In Chapter 3, we present the results related to fault-tolerant consensus in directed graphs using general algorithms. We consider both f-total Byzantine and f-total crash fault models. This Chapter explores the problem in both synchronous and asynchronous systems. Work presented in this Chapter has resulted in the conference publication [83] and technical reports [76, 80].
- In Chapters 4, 5, 6, and 7, we explore the problem of achieving fault-tolerant approximate consensus in directed graphs using iterative algorithms. We consider different fault models, including f-total Byzantine fault model (Chapter 4), generalized Byzantine fault model (Chapter 5), transient Byzantine link fault model (Chapter 6), and f-total crash fault model (Chapter 7). Chapters 4, 5 and 6 consider synchronous systems; whereas, Chapters 4 and 7 also consider asynchronous systems. Work presented in these Chapters has resulted in conference publications [79, 89, 81] and technical reports [84, 82, 90, 91].
- In Chapter 8, we present the results related to reliable broadcast using certified propagation algorithm. We consider f-local Byzantine fault model in this Chapter. Work presented in this Chapter has resulted in the journal publication [75] and technical report [85].
- In Chapter 9, we propose the convex hull consensus problem, and provide an iterative algorithm that tolerates crash faults in asynchronous systems. Work presented in this Chapter has resulted in the conference publication [79] and technical report [78].
- In Chapter 10, we conclude the dissertation and point out future research directions.

Chapter 2

Related Work

This Chapter discusses work related to consensus problem in message-passing network. Since Pease, Shostak, and Lamport posed the *fault-tolerant consensus* problem [60], it has received significant attention over the past three decades [5, 52]. Below, we present relevant papers in the literature and compare this dissertation with closely related work.

2.1 Consensus with Different Assumptions on Graphs

The fault-tolerant consensus problem has been studied extensively in complete networks (e.g., [60, 45, 5, 52]) and in undirected networks (e.g., [32, 27]). However, these conditions are not adequate to fully characterize the *directed* graphs in which fault-tolerant consensus is feasible. Moreover, as we will see later, the algorithms for directed graphs presented in this dissertation are substantially different from those previously developed for undirected graphs. Previous work also studies graph properties for other related problems. Bansal et al. [8] identified tight conditions for achieving exact Byzantine consensus with *authentication* tools in *undirected* graphs. Bansal et al. discovered that all-pair reliable communication is not necessary to achieve consensus when using authentication. Our work differs from Bansal et al. in that our results apply in the absence of authentication or any other security primitives; also our results apply to *directed* graphs. We show that even in the absence of authentication, all-pair reliable communication is not necessary for Byzantine consensus (more details can be found in Chapter 3.7). Alchieri et al. [3] explored the problem of achieving exact consensus in *unknown* networks with Byzantine nodes, but the underlying communication graph is assumed to be *fully-connected* (complete network). In our work related to global algorithms, the network is assumed to be known to all nodes, and when we consider iterative algorithms, nodes only have knowledge of the one-hop neighbors (both incoming and outgoing neighbors), and do not need to learn the network topology. Moreover, we consider arbitrary directed graphs that may be incomplete.

Recently, researchers have explored consensus problem in directed dynamic networks [13, 12, 17, 18, 69], where communication network changes over time. For synchronous systems, [17, 18] solved *approximate* crash-tolerant consensus in directed dynamic networks using local averaging algorithms – whereas, in the *static* networks, we solve (i) *exact* consensus for synchronous systems in directed networks (Chapter 3), and (ii) approximate consensus under different fault models (Chapters 4, 5, 6, and 7). which require different types of algorithms. In the asynchronous setting, [17, 18] addressed approximate consensus with crash faults in *complete* graphs (which are necessarily undirected) – whereas, we solve approximate consensus in asynchronous systems using both general algorithms and iterative algorithms in *static directed* graphs, in which the graph may be incomplete. We also consider Byzantine faults, which is not considered in [13, 12, 17, 18, 69].

[13, 69, 12] considered the *message adversary*, which controls the communication pattern, i.e., the adversary has the power to specify the sets of communication graphs. Biely et al. studied the exact consensus problem [12] and k-set consensus problem [13, 69] (i.e., at most k different outputs at fault-free nodes) in dynamic networks under *message adversary*, and the system is assumed to be synchronous. All the nodes are assumed to be fault-free in [13, 69, 12]. No message is tampered in message adversary model; hence, it is different from transient Byzantine link fault model considered in Chapter 6.

2.2 Iterative Approximate Consensus in Incomplete Graphs

There is also rich work on using iterative algorithms to solve approximate consensus in the presence of faults. Dolev et al. presented the early results on Byzantine fault-tolerant iterative consensus [29]. The initial algorithms [29, 52] were proved correct in *fully connected* networks (i.e., complete networks). Fekete [31] studied the convergence rate of approximate consensus algorithms. Abraham et al. proposed an algorithm for approximate Byzantine consensus [2] that has optimal resilience (optimal number of nodes for achieving consensus).

There have been attempts at achieving approximate fault-tolerant consensus iteratively in *partially* connected graphs. Kieckhafer and Azadmanesh examined the necessary conditions in order to achieve "local convergence" in synchronous [41] and asynchronous [7] systems. Azadmanesh and Bajwa [6] presented a specific class of networks in which convergence condition can be satisfied using iterative algorithms. However, [7, 41, 6] did not identified the tight condition of the communication networks.

A restricted fault model – called "malicious" fault model – in which the faulty nodes are restricted to sending identical messages to their neighbors has also been explored extensively [47, 48, 94, 49]. In contrast, our Byzantine model allows a faulty node to send different messages to different neighbors. LeBlanc and Koutsoukos [47] addressed a continuous time version of the consensus problem with malicious faults in complete graphs. Under both malicious and Byzantine fault models, LeBlanc and Koutsoukos [48] have identified some sufficient conditions under which the continuous time version of iterative consensus can be achieved with up to f faults in the network; however, these sufficient conditions are *not* tight.

For the malicious fault model, LeBlanc et al. [49] have obtained *tight* necessary and sufficient conditions for tolerating up to f faults in the network (f-total fault model). Under the malicious fault model, since a faulty node must send identical messages to all the neighbors, the necessary and sufficient conditions are weaker than those developed for the Byzantine fault model (in Chapter 4). For instance, under the malicious model, iterative consensus is possible in a complete graph consisting of $2f+1$ nodes, whereas at least $3f+1$ nodes are necessary for consensus under the Byzantine fault model.

Iterative approximate consensus algorithms that do not tolerate faulty behavior have been studied extensively in the decentralized control area. Bertsekas and Tsitsiklis [9] and Jadbabaei, Lin and Morse [38] have explored approximate consensus in the absence of faults, using only near-neighbor communication in systems wherein the communication graph may be partially connected and dynamic. The proof technique used for proving correctness of our iterative algorithms (Chapters 5, 6, 7) is inspired by the prior work on non-fault-tolerant iterative algorithms [9, 38].

2.3 Consensus in the Presence of Link Faults

Much effort has also been devoted to the problem of achieving consensus in the presence of link failures [19, 14, 64, 65, 67]. Charron-Bost and Schiper proposed the HO (Heard-Of) model that captures both the link and node failures at the same time [19]. However, the failures are assumed to be benign in the sense that no corrupted message will ever be received in the network. Santoro and Widmayer proposed the *transient* Byzantine link failure model: a different set of links can be faulty at different time [64, 65]. The nodes are assumed to be fault-free in the model. They characterized a necessary condition and a sufficient condition for undirected networks to achieve consensus in the transient link failure model; however, the necessary and sufficient conditions do not match: the necessary and sufficient conditions are specified in terms of node degree and edge-connectivity,[1] respectively. Subsequently, Biely et al. proposed another link failure model that imposes an upper bound on the number of faulty links incident to each node [14]. As a result, it is possible to tolerate $O(n^2)$ link failures with n nodes in the new model. Under this model, Schmid et al. proved lower bounds on number of nodes, and number of rounds for achieving consensus [67]. However, incomplete graphs were not considered in [14, 67].

[1] A graph $G = (\mathcal{V}, \mathcal{E})$ is said to be k-edge connected, if $G' = (\mathcal{V}, \mathcal{E} - X)$ is connected for all $X \subseteq \mathcal{E}$ such that $|X| < k$.

For exact consensus problem, it has been shown that (i) an undirected graph of $2f+1$ node-connectivity[2] is able to tolerate f Byzantine nodes [32]; and (ii) an undirected graph of $2f+1$ edge-connectivity is able to tolerate f Byzantine links [65]. Researchers also showed that $2f+1$ node-connectivity is both necessary and sufficient for the problem of information dissemination in the presence of either f faulty nodes [73] or f *fixed* faulty links [74].[3] Link failures have also been addressed in other contexts, such as distributed method for wireless control network [58], reliable transmission over packet network [51], and estimation over noisy links [66]. These papers considered different problem formulations than the one we consider in Chapter 6.

2.4 Reliable Communication and Broadcast

Several papers have also addressed communication between a single source-receiver pair, i.e., reliable communication problem. Dolev et al. [28] studied the problem of secure communication, which achieves both fault-tolerance and perfect secrecy between a single source-receiver pair in undirected graphs, in the presence of node and link failures. Desmedt and Wang considered the same problem in directed graphs [26]. Shankar et al. [71] investigated reliable communication between a source-receiver pair in directed graphs allowing for an arbitrarily small error probability in the presence of a Byzantine failures. Maurer et al. explored the problem in directed dynamic graphs [53]. In this dissertation, we do not consider secrecy, and address the *consensus* problem rather than the single source-receiver pair problem. Moreover, our work addresses deterministically correct algorithms.

There has also been work on reliable broadcast problem, in which a fault-free source needs to transmit an input to all the other fault-free nodes in the f-local Byzantine fault model. Recall that in this fault model, up to f incoming neighbors of each fault-free nodes may become Byzantine faulty. [42] studied the problem in an infinite grid. [10, 11] developed a sufficient condition in the context of arbitrary network topologies, but the sufficient condition proposed is not tight. [61] provided necessary and sufficient conditions, but the two conditions are not tight either. [37] provided another condition that can approximate (within a factor of 2) the largest f for solving reliable broadcast. Independently, [57] presented the tight condition in *undirected* graphs. The condition presented in [57] is a special case of our condition presented in Chapter 8. In other words, for undirected graphs, the condition in [57] is equivalent to the condition in Chapter 8.

[2] A graph $G = (\mathcal{V}, \mathcal{E})$ is said to be k-node connected, if $G' = (\mathcal{V} - X, \mathcal{E})$ is connected for all $X \subseteq \mathcal{V}$ such that $|X| < k$.

[3] Unlike the "transient" failures in our model, the faulty links are assumed to be fixed throughout the execution of the algorithm in [74].

2.5 Consensus with High-Dimensional Input/Output

In complete networks, the recent papers by Mendes and Herlihy [54] and Vaidya and Garg [88] addressed approximate vector consensus in the presence of Byzantine faults. These papers yielded lower bounds on the number of nodes, and algorithms with optimal resilience for asynchronous [54, 88] as well as synchronous systems [88]. Subsequent work [87] has explored the vector consensus problem in incomplete graphs. These papers are different from our work in Chapter 9 because in our formulation, fault-free nodes have to agree on "largest possible" polytope in the d-dimensional space ($d \geq 1$) that may not necessarily equal to a d-dimensional vector (a single point).

Herlihy et al. [35] introduced the problem of Barycentric agreement (BA). Barycentric agreement has some similarity to convex hull consensus (discussed in Chapter 9), in that the output of Barycentric agreement is not limited to a single value (or a single point). However, the correctness conditions and assumptions on possible input values for Barycentric agreement (BA) are different from those of our convex hull consensus problem, including (i) For a given d-dimensional space, the inputs for BA are vertices of some simplex; whereas, the inputs for convex hull consensus are arbitrary points; (ii) the outputs for BA are sets of vertices of a simplex; whereas, the outputs in our case are convex polytopes (sets of points); and (iii) the agreement property for BA requires containment (i.e., sets of output vertices at fault-free nodes are totally ordered by containment); whereas, convex hull consensus adopts the "approximate" notion, i.e., the distance between the outputs at any pair of fault-free nodes is within ϵ ($\epsilon > 0$).

Chapter 3

Fault-tolerant Consensus Under f-total Faults

3.1 Introduction

The goal of this Chapter is to characterize the necessary and sufficient conditions on the underlying communication graph for solving exact and approximate consensus, respectively, in synchronous and asynchronous systems in the presence of *crash* and *Byzantine node* faults. Here, we consider f-total fault model, which assumes that up to f nodes in the system may be faulty, and all the links are assumed to be reliable. Please refer to Chapter 1.2 for details of our system and fault models. Our problem formulation allows nodes to have complete knowledge of the communication graph. In other words, we assume the usage of general algorithms. The results presented in this Chapter are published in [83].

Table 3.1 identifies the results presented in this Chapter. Where prior work has already obtained the necessary and sufficient conditions, those conditions are also enumerated. Recall that n is the number of nodes in the system. The table on the left is for consensus with up to f crash faults, whereas the table on the right is for consensus with up to f Byzantine faults. The term *connectivity* is used here to mean *node connectivity.* We will often use the terms *graph* and *network* interchangeably.

The problem of asynchronous Byzantine consensus in directed graphs remains open. To prove that the necessary conditions presented in this Chapter are also sufficient, we have developed algorithms that achieve consensus in graphs satisfying those conditions. It turns out that the algorithms required for *directed* graphs

Table 3.1: Summary of tight conditions using general algorithms

Crash-Tolerant Consensus		
	Synchronous	Asynchronous
Undirected graph	$f+1$- connectivity, $n > f$ (follow from well-known results [52, 5])	$f+1$- connectivity, $n > 2f$ (follow from well-known results [52, 5])
Directed graph	*This Chapter*	*This Chapter*

Byzantine Consensus		
	Synchronous	Asynchronous
Undirected graph	$2f+1$- connectivity, $n > 3f$ [32, 27]	$2f+1$- connectivity, $n > 3f$ (follows from [2, 32])
Directed graph	*This Chapter*	Open problem

are substantially different from those previously developed for *undirected* graphs, e.g., [32, 5, 27].

As noted above, our problem formulation allows nodes to have complete knowledge of the network topology. It is also possible to achieve consensus when nodes are *constrained* to only have knowledge of the *local* network topology, particularly their immediate neighbors. We note, however, that the algorithms using only the *local* neighborhood information require richer communication graphs. Such algorithms (iterative algorithms) have been a topic of prior work [17, 9, 38, 49, 18], including our own work [84, 89, 72]. We will present our work on iterative algorithms in Chapters 4, 5, 6 and 7.

3.2 Terminology

Before presenting our results, we introduce some terminology to facilitate the discussion of this Chapter. Upper case letters are used to name sets. Lower case italic letters are used to name nodes. All paths used in our discussion are directed paths. Recall that we consider the directed graph $G(\mathcal{V}, \mathcal{E})$. In G, node j is said to be an incoming neighbor of node i if $(j, i) \in \mathcal{E}$. Let N_i^- be the set of incoming neighbors of node i, i.e., $N_i^- = \{j \mid (j, i) \in \mathcal{E}\}$. Define N_i^+ as the set of outgoing neighbors of node i, i.e., $N_i^+ = \{j \mid (i, j) \in \mathcal{E}\}$. For set $B \subseteq \mathcal{V}$, node i is said to be an incoming neighbor of set B if $i \notin B$, and there exists $j \in B$ such that $(i, j) \in \mathcal{E}$. Given subsets of nodes A and B, set B is said to have k incoming neighbors in set A if A contains k distinct incoming neighbors of B.

Definition 1 *Given disjoint non-empty subsets of nodes A and B, $A \xmapsto{x} B$ if B has at least x distinct incoming neighbors in A. When it is not true that $A \xmapsto{x} B$, we will denote that fact by $A \overset{x}{\not\longmapsto} B$.*

Consider the network in Figure 3.1, which contains two cliques K_1 and K_2, each consisting of 7 nodes. Within each clique, each node has a directed link to the other 6 nodes in that clique – these links within each clique are not shown in the figure. There are 8 directed links with one endpoint in clique K_1 and the other endpoint in clique K_2. In the network, K_2 has 4 incoming neighbors in K_1, namely u_1, u_2, u_3 and u_4. Thus, $K_1 \xmapsto{4} K_2$. Similarly, $K_2 \xmapsto{4} K_1$.

3.3 Main Results of Chapter 3

The main contribution of this Chapter is to identify necessary and sufficient conditions on the underlying communication networks $G(\mathcal{V}, \mathcal{E})$ for achieving fault-tolerant consensus in directed graphs. We summarize the main results in three theorems below. Each theorem below requires the graph to satisfy a certain

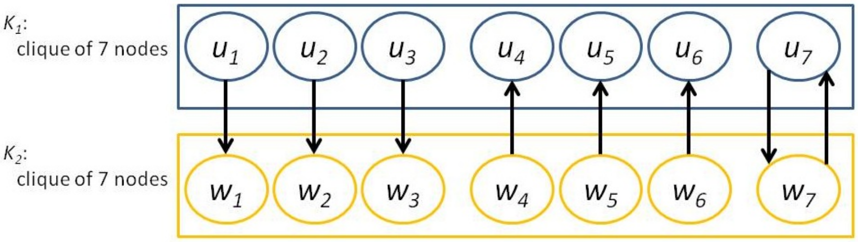

Figure 3.1: Edges within cliques K_1 and K_2 are not shown.

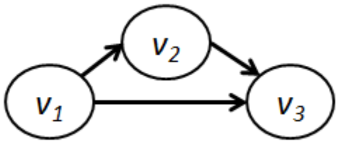

Figure 3.2: Another example.

condition: we name the conditions presented in Theorems 1, 2 and 3 as **CCS** (abbreviating Crash-Consensus-Synchronous), **CCA** (Crash-Consensus-Asynchronous) and **BCS** (Byzantine-Consensus-Synchronous), respectively. Characterization of the necessary and sufficient condition for approximate Byzantine consensus in asynchronous systems remains open.

The precise *validity* conditions that need to be satisfied for different versions of the consensus problem are specified at the start of Chapters 3.4, 3.5 and 3.6, respectively. In brief, for Theorem 1, the output at the nodes must equal the input of one of the nodes. For Theorem 2, the output must be in the range of the inputs of all the nodes. For Theorem 3, the validity condition depends on whether the inputs are binary or not: for binary inputs, the output must be the input of a fault-free node, whereas for multi-valued inputs, the output must equal the input of fault-free nodes when they all have the same input. These conditions are standard in the literature [52, 5].

Theorem 1 *Exact crash-tolerant consensus in a synchronous system is feasible iff for any partition*[1] F, L, C, R *of* $\mathcal{V}$*, where* L *and* R *are both non-empty, and* $|F| \leq f$*, either* $L \cup C \xmapsto{1} R$ *or* $R \cup C \xmapsto{1} L$*.*
(Condition CCS)

Theorem 2 *Approximate crash-tolerant consensus in an asynchronous system is feasible iff for any partition* L, C, R *of* $\mathcal{V}$*, where* L *and* R *are both non-empty, either* $L \cup C \xmapsto{f+1} R$ *or* $R \cup C \xmapsto{f+1} L$*.*
(Condition CCA)

Theorem 3 *Exact Byzantine consensus in a synchronous system is feasible iff for any partition* F, L, C, R *of* $\mathcal{V}$*, where* L *and* R *are both non-empty, and* $|F| \leq f$*, either* $L \cup C \xmapsto{f+1} R$ *or* $R \cup C \xmapsto{f+1} L$*.*
(Condition BCS)

The network shown in Figure 3.1 above satisfies Condition BCS for $f = 2$, whereas the network in Figure 3.2 above satisfies Condition CCS for $f = 1$.

[1] Sets $X_1, X_2, X_3, ..., X_p$ are said to form a partition of set X provided that (i) $\cup_{1 \leq i \leq p} X_i = X$, and (ii) $X_i \cap X_j = \emptyset$ if $i \neq j$.

Intuition For consensus to be achieved, there must be a way for information to "flow between" different subsets of fault-free nodes (subsets L and R in the theorems above), despite the presence of faulty nodes. The different conditions above capture this intuition. Observe that, in each case, for different values of x, we obtain the requirement of the form "either $L \cup C \overset{x}{\longmapsto} R$ or $R \cup C \overset{x}{\longmapsto} L$". Intuitively, information must "flow" either from $L \cup C$ to R, or from $R \cup C$ to L, but it is not necessary that the information flows in both directions – this "asymmetry" in the necessary and sufficient condition is a consequence of the directed nature of the communication network. Note that in Condition CCA (in Theorem 2), the partition does not need set F, unlike Conditions CCS and BCS for the synchronous case.

Lower Bounds on n As shown below, the tight conditions imply the lower bounds on the number of nodes, n. Recall that by definition, $|\mathcal{V}| = n$. These lower bounds are well-known results in complete and undirected graphs [5, 52]. We include them here for completeness. Recall that by assumption, up to f nodes may be faulty in the system.

- Condition CCA implies that $n \geq 2f + 1$ (Lemma 7).
- Condition BCS implies that $n \geq 3f + 1$ (Lemma 11).

We do not include the implication of Condition CCS, since $f + 1$ is a trivial lower bound for crash-tolerant consensus in synchronous systems.

3.4 Exact Crash-tolerant Consensus in Synchronous Systems

An exact crash-tolerant consensus algorithm must satisfy the following three properties: (i) **Agreement**: the output (i.e., decision) at all the fault-free nodes is identical. (ii) **Validity**: the output of each fault-free node equals the input of one of the nodes. (iii) **Termination**: every fault-free node decides on an output in finite amount of time.

Theorem 1 in Chapter 3.3 presents the necessary and sufficient condition (named Condition CCS) for solving the above problem in directed graphs.

3.4.1 Necessity of Condition CCS

It is straightforward why Condition CCS is necessary, since to achieve exact consensus, some information needs to "flow" between two subsets of nodes after some nodes crash. The lemma below formally shows the necessity of Condition CCS using this intuition.

Lemma 1 *Condition CCS is necessary for exact crash-tolerant consensus in a synchronous system.*

Proof: The proof is by contradiction. Suppose that there exists a consensus algorithm for graph $G(\mathcal{V}, \mathcal{E})$, but $G(\mathcal{V}, \mathcal{E})$ does not satisfy the condition in the lemma. Thus, there exists a node partition F, L, C, R of $\mathcal{V}$ such that (i) $|F| \leq f$, (ii) L and R are both non-empty, and (iii) $L \cup C \overset{1}{\not\longmapsto} R$ and $R \cup C \overset{1}{\not\longmapsto} L$. The last two assumptions imply that nodes in $L \cup C$ have no links to nodes in R, and nodes in $R \cup C$ have no links to nodes in L.

Now, consider an execution of the consensus algorithm where F is the set of faulty nodes which crash before the start of the algorithm. All the other nodes are fault-free. This is possible, since by assumption, $|F| \leq f$. Also, suppose that all the nodes in L have input 0, and all the nodes in R have input 1. Nodes in C may have input either 0 or 1.

Consider any node $x \in L$. Since nodes in F fail before taking any steps in the algorithm, and as noted above, there are no links from $C \cup R$ to any node in L, the only input value learned by x throughout the execution of the algorithm is 0. Then to satisfy the validity property, 0 must be the output of node x. Similarly, any node y in R can only learn input values 1 throughout the execution of the algorithm, and thus, 1 must be the output of node y. Since L and R are non-empty and consist of fault-free nodes, the above observations imply that the agreement property is violated. This is a contradiction. □

3.4.2 Sufficiency of Condition CCS

We prove the sufficiency of Condition CCS constructively by presenting an algorithm, called MVC, and proving its correctness. Algorithm MVC can achieve consensus with multi-valued inputs. MVC uses Algorithm Min-Max presented below as a component. Algorithm Min-Max can achieve consensus with *binary* inputs (0 or 1). The proposed algorithms prove sufficiency of Condition CCS, but they are not necessarily the most efficient. Development of optimal algorithms needs further research.

Binary Consensus

Note that Algorithm *Min-Max* has input parameter x_i. To achieve binary consensus, each node i performs Algorithm *Min-Max* passing its binary input value as parameter x_i to Algorithm *Min-Max*. Algorithm Min-Max uses *Compute* as a sub-routine. *Compute* has two parameters: t, which is a binary value, and *Function*, which may be specified as *Min* and *Max*. In the last step of each round in *Compute* at node i, the *Function* is applied to set S_i. $Min(S_i)$ returns minimum of the values in set S_i, and $Max(S_i)$ returns the maximum of the values in set S_i.

Algorithm Min-Max(x_i) for node $i \in \mathcal{V}$

```
*Initialization*: $v_i[0]$ := parameter $x_i$ passed to Min-Max

• For phase number $p := 1$ to $2f + 2$:
        If $p \ mod \ 2 = 0$, then                                  (Min Phase)
            $v_i[p]$ := Compute ($v_i[p-1]$, Min)
        Else,                                                       (Max Phase)
            $v_i[p]$ := Compute ($v_i[p-1]$, Max)

• Return $v_i[2f + 2]$
```

Compute(t, *Function*) for node $i \in \mathcal{V}$

```
• $\tau_i := t$

• Perform $n - 1$ rounds, each round consisting of the four steps below:
        Send $\tau_i$ to all the nodes in $N_i^+ \cup \{i\}$
        Receive values from $N_i^- \cup \{i\}$
        Denote the set of values received
            in the previous step as $S_i$
        $\tau_i := Function(S_i)$

• Return $\tau_i$
```

Correctness of Algorithm Min-Max with Binary Inputs We first prove an useful lemma, and introduce the notion of *source* of the graph.

Lemma 2 *Suppose that graph $G(\mathcal{V}, \mathcal{E})$ satisfies Condition CCS. For any $F \subseteq \mathcal{V}$, such that $|F| \leq f$, let G_F denote the subgraph of G induced by the nodes in $\mathcal{V} - F$. There exists at least one node in G_F that has directed paths in G_F to all nodes in $\mathcal{V} - F$. Such a node is said to be a source for G_F.*

Proof: The proof of the lemma is by contradiction. Suppose that Graph $G(\mathcal{V}, \mathcal{E})$ satisfies Condition CCS, and for some $F \subseteq \mathcal{V}$, $|F| \leq f$, there exists a pair of nodes $i, j \notin F$ such that there is no node s that has directed paths to both i and j in subgraph G_F induced by nodes in $\mathcal{V} - F$. For the subgraph G_F and a node x in $\mathcal{V} - F$, define S_x as the set of all nodes that have directed paths in G_F to node x. Note that S_x contains x as well, because x trivially has a path to itself.

By assumption, S_i and S_j are disjoint. Moreover, there must be no path from any node in S_i to any node in S_j in G_F, and vice versa, since otherwise, there would exist some node that can reach both nodes i and j, which contradicts our assumption above. Now, define L, C, R as follows:

- $L := S_i$
- $R := S_j$
- $C := \mathcal{V} - F - L - R$

Then, we make the following observations:

- *F and C may be empty, but L and R are non-empty*: This is true because $i \in S_i = L$ and $j \in S_j = R$.
- *Nodes in C (if non-empty) have no link to nodes in $L \cup R$*: If some node $c \in C$ has a link to some node $x \in L = S_i$, then c will be able to reach node i on a directed path via node x (since $x \in S_i$ has a path to i, by definition of set S_i). This would then imply that c must be in S_i, however, that contradicts the definition of C as $\mathcal{V} - F - L - R$. By a similar argument, nodes in C cannot have links to nodes in R.
- *There is no link from any node in L to nodes in R, and vice versa*: Recall that $L = S_i$ and $R = S_j$. If some node $x \in L$ has a link to a node $y \in R$, then x will have a directed path to node j via node y. However, this contradicts our assumption above that no node has directed paths to both i and j.

These observations together imply that $L \cup C \overset{1}{\not\longrightarrow} R$ and $C \cup R \overset{1}{\not\longrightarrow} L$. That is, $G(\mathcal{V}, \mathcal{E})$ does not satisfy Condition CCS. This is a contradiction. Thus, Lemma 2 is proved. □

The lemma defines the notion of a *source* node. Essentially, the lemma shows the existence of a *directed rooted spanning tree* in the induced graph G_F, which has the source node as the root. Presence of such "source" nodes (or root) is crucial in achieving consensus. Similar observations were first made in the context of fault-free consensus [9, 38], and also in the context of other versions of fault-tolerant consensus problems [17, 89, 84, 12] (although the exact manner in which the source node is identified differs for the different problems). The key distinguishing feature of the results presented in this Chapter is in the algorithms developed to solve the consensus problems. For instance, the structure of the above Min-Max algorithm – making alternating use of Min and Max function – has not been applied for consensus previously, to the best of our knowledge.

Now, we are ready to show the correctness of Algorithm Min-Max. The proof of correctness assumes that graph $G(\mathcal{V}, \mathcal{E})$ satisfies Condition CCS.

Lemma 3 *Algorithm Min-Max satisfies termination, agreement and validity properties.*

Proof:

Since Algorithm Min-Max executes a fixed number of phases, its termination occurs in finite time. Validity is satisfied trivially as well. Now we prove that the algorithm satisfies the agreement property when the inputs are binary (0 or 1). We start by observing that $Compute(t, Min)$ never returns a value larger than parameter t passed to *Compute*, and $Compute(t, Max)$ never returns a value smaller than parameter t passed to *Compute*.

Fix an execution of the algorithm. Since there are $2f + 2$ phases. There must exist a pair of consecutive phases p^*, $p^* + 1$ such that *no node crashes* in phases p^* and $p^* + 1$. Without loss of generality, let p^* be the Min Phase (i.e., p^* *mod* $2 = 0$) and $p^* + 1$ be the Max Phase. Denote by F the set of nodes that crash before starting phase p^*.

Lemma 2 shows the existence of a source node that has directed paths in G_F to all nodes in $\mathcal{V} - F$ (G_F is defined in the Lemma). In general, there may be multiple such source nodes in G_F. Consider the two cases below. In each case, we show that agreement is achieved.

- *Case I: There exists a source s in G_F for which $v_s[p^* - 1] = 0$:* Thus, during the Min Phase p^*, node s will call *Compute(0,Min)*. Then during the first round of *Compute* in phase p^*, those nodes in $\mathcal{V} - F$ with incoming links from node s will update their τ variable (within *Compute*) to be 0. Recall that we are presently assuming binary inputs. Since the source node has directed paths (of length at most $n - 1$) to all the nodes in G_F, it follows by induction that each node i in $\mathcal{V} - F$ will update its state τ_i to be 0 by the end of the $n - 1$ rounds performed within *Compute*. Thus, when *Compute* returns, $v_i[p^*]$ at each $i \in \mathcal{V} - F$ will be set to 0. It should be easy to see that the remaining phases will not change the value of v_i at the fault-free nodes, ensuring agreement when the algorithm terminates.

- *Case II: For each source s in G_F, $v_s[p^* - 1] = 1$:*

  In this case, we argue that, for each source s, $v_s[p^*] = 1$. Suppose, by way of contradiction, that each source s of G_F has $v_s[p^* - 1] = 1$, but there exists a source node s' for which $v_{s'}[p^*] = 0$. For this to happen, node s' must receive 0 on a path from some other *non-source* node z during phase p^*. This implies that $v_z[p^* - 1] = 0$; additionally, the fact that there exists a path in G_F from z to the source node s' implies that z is also a source in G_F. This contradicts the assumption that all source nodes in G_F have state equal to 1 at the start of phase p^*.

  This shows that, for each source node s, we have $v_s[p^*] = 1$. Now consider Max Phase $p^* + 1$. Recall that no node crashes in Phases p^* and $p^* + 1$. Thus, by an argument analogous to that used for Min

Phase p^* in Case I above, it follows that, for all $i \in \mathcal{V} - F$, $v_i[p^* + 1] = 1$, achieving agreement. Any additional phases beyond phase $p^* + 1$ will not result in violation of the agreement, similar to Case I.

□

Multi-Valued Consensus

Algorithm Min-Max above is proven correct for binary inputs. We will prove that the Algorithm MVC presented below performs exact consensus with inputs being some integer in the range $[0, K]$, where $K \geq 1$. Algorithm MVC uses Algorithm *Min-Max* as well as *Compute* defined above.

Algorithm MVC for node i

```
Initialization: $w_i[0]$ := input of node $i$

Repeat for $l := 0$ to $K$

  1. $w_i[l+1] := Compute(w_i[l], Max)$
  2. if $w_i[l] = l$, then $y_i[l] := 0$; otherwise, $y_i[l] := 1$
  3. if Min-Max($y_i[l]$) returns 0, then
         terminate with output $l$
```

Step 3 of the algorithm performs Algorithm Min-Max, with node i passing $y_i[l]$ as the parameter to Min-Max. The "if" statement in step 3 checks the value returned by Algorithm Min-Max, and terminates the algorithm if the returned value is 0. Recall that the system is synchronous. Thus, all fault-free nodes perform *Min-Max* in step 3 in iteration l of the algorithm synchronously. Similarly, all fault-free nodes perform *Compute* (in step 1) in iteration l of the algorithm synchronously.

Correctness of Algorithm MVC The proof assumes that graph $G(\mathcal{V}, \mathcal{E})$ satisfies Condition CCS. Observe that Algorithm MVC specifies up to $K + 1$ iterations, each iteration consisting of 3 steps enumerated in the algorithm. However, the algorithm may potentially terminate before completing all $K + 1$ iterations (due to the check in step 3).

Since Min-Max is a binary consensus algorithm, each node that completes execution of Algorithm Min-Max in iteration l of MVC (i.e., without crashing before this instance of Min-Max) will obtain the same return value from Min-Max in iteration l. This ensures that all fault-free nodes terminate after completing an identical number of iterations. Now we present a lemma that is useful to prove correctness of Algorithm MVC.

Lemma 4 *If a node i initiates iteration l of Algorithm MVC ($0 \le l \le K$), then (i) $w_i[l] \ge l$, and (ii) $w_i[l]$ equals the input of some node in the system.*

Proof: The proof is by induction. Consider $l = 0$. $w_i[0]$ is initialized to be the input of node i. All inputs are in the range $[0, K]$, which implies that $w_i[0] \ge 0$. Thus, both parts of the statement of the lemma are true for $l = 0$.

Now assume that the statement of the lemma is true for some l, where $0 \le l < K$. Thus, for each node i that initiates iteration l, $w_i[l] \ge l$ and $w_i[l]$ is the input of some node in the system. Now, if Algorithm MVC terminates without performing iteration $l + 1$, the proof of the lemma is complete. Therefore, let us now consider the case that Algorithm MVC is not terminated after completing l iterations. This implies that Min-Max($y_i[l]$) returns 1 in Step 3 of iteration l. Since Algorithm Min-Max is a binary consensus algorithm, the validity condition and the fact that it returns 1 implies that there exists some node j with $y_j[l] = 1$ such that (i) j did not crash before starting the execution of Min-Max($y_j[l]$) in step 3 of iteration l at node j, and (ii) node i has a path from node j consisting only of nodes that have *not* crashed before initiating step 3 of iteration l. Due to the rules for setting the value of $y_j[l]$ in step 2, and the fact that $y_j[l] = 1$, we can infer that that $w_j[l] > l$.

Finally, all nodes that are fault-free at least until initiating step 3 in iteration l must also be fault-free until they finish performing step 1 in iteration l. This implies that when *Compute* is performed in step 1 of iteration l, there exists a path from node j to node i. Existence of this path then ensures that, at node i, *Compute* returns a value $\ge w_j[l] > l$. Thus, $w_i[l + 1] > l$. That is, $w_i[l + 1] \ge l + 1$.

Secondly, *Compute* at node i in iteration l will only return a value that is passed as the first parameter to *Compute* by at least one of the nodes initiating the execution of *Compute* in iteration l. Thus, $w_i[l + 1]$ must equal $w_j[l]$ for some node j – by induction basis, $w_j[l]$ must be the input of some node in the system, and therefore, $w_i[l + 1]$ also equals the input of that node. This concludes the proof. □

Lemma 5 *Algorithm MVC satisfies termination, agreement and validity properties.*

Proof: Termination in finite time occurs because Algorithm *Min-Max* and *Compute* return in finite time, and Algorithm MVC performs a finite number of iterations. As discussed earlier, all fault-free nodes terminate after completing an *identical* number of iterations. There are two cases to consider depending on the number of iterations completed before the algorithm is terminated.

Case 1: Algorithm MVC terminates only after completing the last iteration with $l = K$. Due to Lemma 4, for each node j that initiates iteration with $l = K$, we have $w_j[K] \geq K$. However, since all inputs are in the range $[0, K]$, and due to the use of *Compute* to update w_j, we have that $w_j[K] \leq K$ as well. This means that $w_j[K] = K$. But by Lemma 4, $w_j[K]$ is also the input of some node in the system; it then follows that some node in the system had value K as the input.

Since $w_j[K] = K$, in iteration $l = K$, $y_j[K] = 0$ for each node j that completes step 2, and thus Min-Max can only return 0 in step 3, resulting in the output value of K at all fault-free nodes. Thus agreement is achieved. As argued above, K is the input of some node, the validity property is also satisfied.

Case 2: Algorithm MVC terminates after completing iteration $l < K$. This implies that all nodes that are fault-free until the end of iteration l must have necessarily obtained the return value of 0 from Min-Max in iteration l, and therefore, they will all terminate with output value l. Thus agreement is achieved. The fact that the return value from Min-Max is 0 implies that at least one node, say node j, must have initiated Min-Max in iteration l with $y_j[l] = 0$, implying (due to step 2) that $w_j[l] = l$. By Lemma 4, $w_j[l] = l$ must be the input of some node. Since l is also the output of the algorithm in this case, validity condition is also satisfied. □

Lemma 5 shows that Condition CCS is sufficient.

3.5 Approximate Crash-tolerant Consensus in Asynchronous Systems

Approximate crash-tolerant consensus must satisfy the following three properties, where $\epsilon > 0$: (i) **ϵ-agreement**: the difference between outputs at any two fault-free nodes is $< \epsilon$. (ii) **Validity**: the output at any fault-free node is within the range of the inputs at all the nodes. (iii) **Termination:** every fault-free node decides on an output in finite amount of time. For simplicity, we assume that the input at each node is some *real number* in the range $[0, K]$. Note that if $K < \epsilon$, then the problem is trivial, so K is assumed to be $\geq \epsilon$.

Theorem 2 in Chapter 3.3 presents the necessary and sufficient condition (named Condition CCA) for solving the above problem in directed graphs.

3.5.1 Necessity of Condition CCA

The proof of the following lemma is different from Lemma 1 even though it follows similar intuition. Here, to achieve approximate consensus, some "information" must be able to flow between two subsets of nodes even if some links may be slow – a result due to asynchrony assumption.

Lemma 6 *Condition CCA is necessary for approximate crash-tolerant consensus in an asynchronous system.*

Proof: The proof is by contradiction. Suppose that there exists a correct approximate consensus algorithm in $G(\mathcal{V}, \mathcal{E})$, but $G(\mathcal{V}, \mathcal{E})$ does not satisfy the condition in the lemma. That is, there exists a node partition L, C, R such that L and R are non-empty, and $L \cup C \overset{f+1}{\not\longrightarrow} R$ and $C \cup R \overset{f+1}{\not\longrightarrow} L$. Let $O(L)$ denote the set of nodes in $C \cup R$ that have outgoing links to nodes in L, i.e., $O(L) = \{i \mid i \in C \cup R, \;\; N_i^+ \cap L \neq \emptyset\}$. Similarly, define $O(R) = \{j \mid j \in L \cup C, \;\; N_j^+ \cap R \neq \emptyset\}$. Since $L \cup C \overset{f+1}{\not\longrightarrow} R$ and $C \cup R \overset{f+1}{\not\longrightarrow} L$, we have $|O(L)| \leq f$ and $|O(R)| \leq f$.

Consider the scenario where (i) each node in L has input 0; (ii) each node in R has input ϵ; (iii) nodes in C (if non-empty) have arbitrary inputs in $[0, \epsilon]$; (iv) no node crashes; and (v) the message delay for communications channels from $O(L)$ to L and from $O(R)$ to R is arbitrarily large compared to all the other channels. Recall that such a scenario is possible, since we have assumed that the input range is $[0, K]$, where $K \geq \epsilon$. Consider nodes in L.

Since messages from the set $O(L)$ take arbitrarily long to arrive at the nodes in L, and $|O(L)| \leq f$, from the perspective of the nodes in L, the nodes in $O(L)$ appear to have crashed. Thus, nodes in L must decide on their output without waiting to hear from the nodes in $O(L)$. Consequently, to satisfy the validity property, the output at each node in L has to be 0, since 0 is the input of all the nodes in L. Similarly, nodes in R must decide their output without hearing from the nodes in $O(R)$; they must choose output as ϵ, because the input at all the nodes in R is ϵ. Thus, the ϵ-agreement property is violated, since the difference between outputs at fault-free nodes is not $< \epsilon$. This is a contradiction. □

The lemma below shows the lower bound on the number of nodes for solving approximate crash-tolerant consensus in asynchronous systems. Recall that $|\mathcal{V}| = n$ by definition.

Lemma 7 *If $G(\mathcal{V}, \mathcal{E})$ satisfies Condition CCA, then $n \geq 2f + 1$.*

Proof: For $f = 0$, the lemma is trivially true, since we assume that $n \geq 2$ as stated in Chapter 1.2. Now consider $f > 0$. The proof is by contradiction. Suppose that $n \leq 2f$. Partition $\mathcal{V}$ into three subsets A, B, F

such that $F = \emptyset$, $0 < |A| \leq f$, and $0 < |B| \leq f$. Such a partition can be found because $2 \leq n \leq 2f$. Since A, B are both non-empty, and contain at most f nodes each, we have $A \overset{f+1}{\not\mapsto} B$ and $B \overset{f+1}{\not\mapsto} A$, violating Condition CCA. This proves the lemma. □

3.5.2 Sufficiency of Condition CCA

We prove the sufficiency of Condition CCA constructively by presenting an algorithm, called Algorithm WA (Wait-and-Average), and proving its correctness. The algorithm, presented below, assumes that each node has the knowledge of the network topology, and the algorithm proceeds in *asynchronous* phases. In each phase, nodes flood messages containing the current value of their state variable v, their identifier, and a phase index. Each node i waits until it has received an "adequate" set of values from other nodes, where "adequate" is made precise by Condition WAIT defined below. Then, node i updates its state variable v to be the *average* of set of values received in the current phase, and then proceeds to the next phase. When node i has finished p_{end} phases, it produces an output that equals the current value of state variable v; p_{end} is an integer $> \log_{n/(n-1)} \frac{K}{\epsilon}$.

In Algorithm WA, observe that $heard_i[p]$ is the set of nodes from which node i has received values during phase p. As seen in the algorithm pseudo-code, node i performs the averaging operation to update its state variable v_i if Condition WAIT below holds for the first time. Algorithm WA is an extension of an approximate consensus algorithm for complete graphs [29]. The key contribution of our work is to identify *Condition WAIT*.

Condition WAIT: For $F_i \subseteq \mathcal{V}$, where $|F_i| \leq f$, denote by $reach_i(F_i)$ the set of nodes that have paths to node i in the subgraph induced by the nodes in $\mathcal{V} - F_i$ (i.e., the paths do not contain nodes in F_i). Condition *WAIT* is satisfied at node i if **there exists** a set $F_i \subseteq \mathcal{V}$, where $|F_i| \leq f$, such that $reach_i(F_i) \subseteq heard_i[p]$. ($reach_i(F_i)$ may be different in each phase, since it depends on the message delays. For simplicity, we ignore the phase index p in the notation.)

Algorithm WA for node $i \in \mathcal{V}$

```
$p_{end}$ is an integer $> \log_{n/(n-1)} \frac{K}{\epsilon}$.

• $v_i[0] :=$ input at node $i$

• For Phase $p := 1$ to $p_{end}$:

    – On entering phase $p$:

        $R_i[p] := \{v_i[p-1]\}$
        $heard_i[p] := \{i\}$
        Send message $(v_i[p-1], i, p)$ to
          all the outgoing neighbors
  – When message $(h, j, p)$ is received for the *first time*:
        $R_i[p] := R_i[p] \cup \{h\}$        // $R_i[p]$ is a multiset
        $heard_i[p] := heard_i[p] \cup \{j\}$
        Send message $(h, j, p)$ to
          all the outgoing neighbors
  – When Condition $WAIT$ holds for the first time in phase $p$:
        $v_i[p] := \frac{\sum_{v \in R_i[p]} v}{|R_i[p]|}$
        If $p < p_{end}$, begin the next phase
        Else,  output $v_i$
```

Note that $R_i[p]$ is a multiset, and thus may contain multiple instances of the same value.

Correctness of Algorithm WA We first prove an useful lemma. Let $heard_i^*[p]$ denote the set $heard_i[p]$ when Condition WAIT holds for the first time at node i in phase p. The correctness of Algorithm WA relies on the following lemma, which assumes graph $G(\mathcal{V}, \mathcal{E})$ satisfies Condition CCA. In a given execution, define $F[p]$ as the nodes that have *not* computed value $v[p]$ in phase p, i.e., nodes in $F[p]$ have crashed before computing $v[p]$.

Lemma 8 *For phase $p \geq 1$, consider two nodes $i, j \in \mathcal{V} - F[p]$. Then, $heard_i^*[p] \cap heard_j^*[p] \neq \emptyset$.*

Proof: By construction, there exist two sets F_i and F_j such that Condition WAIT holds for sets $heard_i^*[p]$ and F_i at node i, and for sets $heard_j^*[p]$ and F_j at node j. In other words, (i) $F_i \subseteq \mathcal{V}$ and $|F_i| \leq f$, (ii) $F_j \subseteq \mathcal{V}$ and $|F_j| \leq f$, (iii) $reach_i(F_i) \subseteq heard_i^*[p]$, and (iv) $reach_j(F_j) \subseteq heard_j^*[p]$. If $reach_i(F_i) \cap reach_j(F_j) \neq \emptyset$, then the proof is complete, since $reach_i(F_i) \subseteq heard_i^*[p]$ and $reach_j(F_j) \subseteq heard_j^*[p]$. Thus, $heard_i^*[p] \cap heard_j^*[p] \neq \emptyset$.

Now, consider the case when $reach_i(F_i) \cap reach_j(F_j) = \emptyset$. We will derive a contradiction in this case. Recall that $reach_i(F_i)$ is defined as the set of nodes that have paths to node i in the subgraph induced by the nodes in $\mathcal{V} - F_i$. This implies that in graph G, the incoming neighbors of set $reach_i(F_i)$ are contained in set F_i. Similarly, in graph G, the incoming neighbors of set $reach_j(F_j)$ are contained in set F_j.

In graph G, we will find subsets of nodes L, C, R that violate Condition CCA. Let $L = reach_i(F_i)$, $R = reach_j(F_j)$ and $C = \mathcal{V} - L - R$. Observe that since $reach_i(F_i) \cap reach_j(F_j) = \emptyset$, L, C, R form a partition of $\mathcal{V}$. Moreover, $i \in reach_i(F_i)$ and $j \in reach_j(F_j)$; hence, $L = reach_i(F_i)$ and $R = reach_j(F_j)$ are both non-empty. Let $N(L)$ be the set of incoming neighbors of set L. By definition, $N(L)$ is contained in $R \cup C$. Since $L = reach_i(F_i)$, the only nodes that may be in $N(L)$ are also in F_i as argued above, i.e., $N(L) \subseteq F_i$. By assumption, $|F_i| \leq f$. Therefore, $|N(L)| \leq f$, which implies that $R \cup C \overset{f+1}{\not\mapsto} L$. Similarly, we can argue that $L \cup C \overset{f+1}{\not\mapsto} R$. These two conditions together show that G violates Condition CCA, a contradiction. Thus, $reach_i(F_i) \cap reach_j(F_j) \neq \emptyset$, which implies that $heard_i^*[p] \cap heard_j^*[p] \neq \emptyset$. This completes the proof. □

Lemma 9 *Algorithm WA satisfies termination, ϵ-agreement and validity properties.*

Proof:

Validity and termination properties are trivially true. Now, we prove that Algorithm WA achieves ϵ-agreement. The methodology in this proof below is borrowed from other related work (e.g., [29, 5, 52, 9]).

Recall that $F[p]$ is defined as the nodes that have *not* computed value $v[p]$ in phase p, i.e., nodes in $F[p]$ have crashed before computing $v[p]$. Thus, $\mathcal{V} - F[p]$ is the set of nodes that complete the computation of $v[p]$.

For $p \geq 1$, define

$$M[p] = \max_{i \in \mathcal{V} - F[p]} v_i[p] \tag{3.1}$$

and

$$m[p] = \min_{i \in \mathcal{V} - F[p]} v_i[p] \tag{3.2}$$

With a slight abuse of terminology, let $M[0]$ and $m[0]$ denote the upper bound and the lower bound on the inputs, respectively. Recall that we have assumed that the input range is $[0, K]$, where $K \geq \epsilon$. Thus,

$$M[0] = K \quad \text{and} \quad m[0] = 0.$$

Define $\psi[p]$ as the maximum difference between states at nodes in $\mathcal{V} - F[p]$ in the end of phase p. Thus,

$$\psi[p] = M[p] - m[p] \tag{3.3}$$

Let us denote by $\|x\|$ the absolute value of a real number x. Then,

$$\psi[p] = \max_{i,j \in \mathcal{V} - F[p]} \|v_i[p] - v_j[p]\| \tag{3.4}$$

By definition of $m[p-1]$ and $M[p-1]$, for each node $k \in \mathcal{V} - F[p-1]$, we have

$$m[p-1] \leq v_k[p-1] \leq M[p-1] \tag{3.5}$$

$$m[p-1] \leq v_k[p-1] \leq M[p-1] \tag{3.6}$$

Now consider nodes $i, j \in \mathcal{V} - F[p]$. The multiset $R_i[p]$ at node i changes when node i receives new messages. Let $R_i^*[p]$ denote the multiset $R_i[p]$ used by node i to compute $v_i[p]$. Similarly, let $R_j^*[p]$ denote the multiset $R_j[p]$ used by node j to compute $v_j[p]$.

Let $r_i = |R_i^*[p]|$, the size of the multiset $R_i^*[p]$. Similarly, let $r_j = |R_i^*[p]|$. For brevity, the notation r_i and r_j does not include the phase index $[p]$.

By Lemma 8, there exists a common value in $R_i^*[p]$ and $R_j^*[p]$. Denote by c the common value. Note that by construction c is a state of some node in $\mathcal{V} - F[p-1]$. Thus, $m[p-1] \leq c \leq M[p-1]$. Define

$$\gamma = \frac{1}{n}.$$

By Algorithm WA, we have

$$\begin{aligned}
v_i[p] &= \sum_{k \in R_i^*[p]} \frac{1}{r_i} v_k[p-1] \\
&\leq \frac{c}{r_i} + (1 - \frac{1}{r_i}) M[p-1] \quad \text{due to (3.1)} \\
&\leq \gamma c + (\frac{1}{r_i} - \gamma) c + (1 - \frac{1}{r_i}) M[p-1] \\
&\leq \gamma c + (1-\gamma) M[p-1] \qquad (3.7)
\end{aligned}$$

The last inequality is because $\gamma \leq \frac{1}{r_i}$ and $c \leq M[p-1]$.

$$\begin{aligned}
v_j[p] &= \sum_{k \in R_j^*[p]} \frac{1}{r_j} v_k[p-1] \\
&\geq \frac{c}{r_j} + (1 - \frac{1}{r_j}) m[p-1] \quad \text{due to (3.2)} \\
&\geq \gamma c + (\frac{1}{r_j} - \gamma) c + (1 - \frac{1}{r_j}) m[p-1] \\
&\geq \gamma c + (1-\gamma) m[p-1] \qquad (3.8)
\end{aligned}$$

The last inequality is because $\gamma \leq \frac{1}{r_j}$ and $c \geq m[p-1]$.

Now, subtracting (3.8) from (3.7), we get

$$v_i[p] - v_j[p] \leq (1-\gamma)(M[p-1] - m[p-1]) \tag{3.9}$$

By swapping the role of i and j above, we can show that

$$v_j[p] - v_i[p] \leq (1-\gamma)(M[p-1] - m[p-1]) \tag{3.10}$$

(3.9) and (3.10) together imply that

$$\begin{aligned} \|v_i[p] - v_j[p]\| &\leq (1-\gamma)(M[p-1] - m[p-1]) \\ &\leq (1-\gamma)\psi[p-1] \qquad \text{due to} \quad (3.3) \end{aligned}$$

Note that the first inequality is because $M[p-1] \geq m[p-1]$. The above inequality holds for each pair of nodes i, j that have computed $v[p]$ in phase p, so we have

$$\max_{i,j \in \mathcal{V}-F[p]} \|v_i[p] - v_j[p]\| \leq (1-\gamma)\psi[p-1]$$

This together with (3.4) implies that

$$M[p] - m[p] \leq (1-\gamma)(M[p-1] - m[p-1]) \tag{3.11}$$

By repeated application of (3.11), we get

$$M[p] - m[p] \leq (1-\gamma)^p (M[0] - m[0]) \tag{3.12}$$

Therefore, for a given $\epsilon > 0$, if

$$p > \log_{1/(1-\gamma)} \frac{M[0] - m[0]}{\epsilon}, \tag{3.13}$$

then

$$M[p] - m[p] < \epsilon \tag{3.14}$$

Recall that we have assumed that the input range is $[0, K]$. Also, $\gamma = \frac{1}{n}$. Then, if we choose

$$p_{end} > \log_{n/(n-1)} \frac{K}{\epsilon},$$

then Algorithm WA satisfies ϵ-agreement property due to (3.13) and (3.14). □

3.6 Exact Byzantine Consensus in Synchronous Systems

This Chapter considers Byzantine faults. An exact Byzantine consensus algorithm with *binary* inputs must satisfy the following properties: (i) **Agreement**: the output (i.e., decision) at all the fault-free nodes is identical. (ii) **Validity**: the output of every fault-free node equals the input of a fault-free node. (iii) **Termination:** every fault-free node decides on an output in finite amount of time. Note that the validity property is slightly different from the one for crash-tolerant consensus. For multi-valued Byzantine consensus, we consider the weaker version of validity (as discussed in Chapter 3.6.6). Theorem 3 in Chapter 3.3 presents the necessary and sufficient condition (named Condition BCS) for solving the above problem in directed graphs. We prove that Condition BCS is necessary in Chapter 3.6.2. Then, we show that Condition BCS is sufficient for Byzantine consensus with binary inputs in Chapter 3.6.5. Finally, we discuss how Condition BCS is also sufficient for Byzantine consensus with multi-valued inputs (and weaker version validity property) in Chapter 3.6.6.

3.6.1 Terminology and Notations

For simplicity of presentation, we introduce the following notations:

In Chapter 3.6, we define $\rightarrow$ and $\not\rightarrow$ to be $\overset{f+1}{\longmapsto}$ and $\overset{f+1}{\not\longmapsto}$, respectively.

Recall that $\overset{f+1}{\longmapsto}$ and $\overset{f+1}{\not\longmapsto}$ are defined in Chapter 3.3. We will use $\rightarrow$ and $\not\rightarrow$ throughout the discussion on Condition BCS (Byzantine Consensus in Synchronous systems), i.e., discussion in Chapter 3.6.

We now describe terminology that is used in this Chapter. Upper case italic letters are used below to name subsets of $\mathcal{V}$, and lower case italic letters are used to name nodes in $\mathcal{V}$.

Incoming neighbors:

- Node i is said to be an incoming neighbor of node j if $(i, j) \in \mathcal{E}$.

- For set $B \subseteq \mathcal{V}$, node i is said to be an incoming neighbor of set B if $i \notin B$, and there exists $j \in B$ such that $(i, j) \in \mathcal{E}$. Set B is said to have k incoming neighbors in set A if set A contains k distinct incoming neighbors of B.

Directed paths: All paths used in our discussion are directed paths.

- Paths from a node i to another node j:
  - For a directed path from node i to node j, node i is said to be the "source node" for the path.
  - An "(i, j)-path" is a directed path from node i to node j. An "(i, j)-path excluding X" is a directed path from node i to node j that does not contain any node from set X.
  - Two paths from node i to node j are said to be "disjoint" if the two paths only have nodes i and j in common, with all remaining nodes being distinct.
  - The phrase "d disjoint (i, j)-paths" refers to d pairwise disjoint paths from node i to node j. The phrase "d disjoint (i, j)-paths excluding X" refers to d pairwise disjoint (i, j)-paths that do not contain any node from set X.
- Every node i trivially has a path to itself. That is, for all $i \in \mathcal{V}$, an (i, i)-path excluding $\mathcal{V} - \{i\}$ exists.
- Paths from a set S to node $j \notin S$:
  - A path is said to be an "(S, j)-path" if it is an (i, j)-path for some $i \in S$. An "(S, j)-path excluding X" is a (S, j)-path that does not contain any node from set X.
  - Two (S, j)-paths are said to be "disjoint" if the two paths only have node j in common, with all remaining nodes being distinct (including the source nodes on the paths).
  - The phrase "d disjoint (S, j)-paths" refers to d pairwise disjoint (S, j)-paths. The phrase "d disjoint (S, j)-paths excluding X" refers to d pairwise disjoint (S, j)-paths that do not contain any node from set X.

Definition 2 *Given disjoint subsets A, B, F of $\mathcal{V}$ such that $|F| \leq f$, set A is said to $\underline{\text{propagate in } \mathcal{V} - F}$ to set B if either (i) $B = \emptyset$, or (ii) for each node $b \in B$, there exist at least $f + 1$ disjoint (A, b)-paths excluding F.*

We will denote the fact that set A propagates in $\mathcal{V} - F$ to set B by the notation

$$A \overset{\mathcal{V}-F}{\rightsquigarrow} B.$$

When it is not true that $A \overset{\mathcal{V}-F}{\rightsquigarrow} B$, we will denote that fact by

$$A \overset{\mathcal{V}-F}{\not\rightsquigarrow} B.$$

Definition 3 *For $F \subset \mathcal{V}$, graph G_{-F} is obtained by removing from $G(\mathcal{V}, \mathcal{E})$ all the nodes in F, and all the links incident on nodes in F.*

Definition 4 *A subgraph S of G is said to be* strongly connected*, if for all nodes i, j in S, there exists an (i, j)-path in G.*

3.6.2 Necessity of Condition BCS

Recall that Condition BCS (Byzantine Consensus in Synchronous systems) is defined in Chapter 3.3. The following lemma proves that Condition BCS is necessary.

Lemma 10 *Exact Byzantine consensus in a synchronous system is feasible* **only if** *Condition BCS holds in $G(\mathcal{V}, \mathcal{E})$, i.e., for any partition F, L, C, R of $\mathcal{V}$, where L and R are both non-empty, and $|F| \le f$, either $L \cup C \to R$ or $R \cup C \to L$.*

Recall that in Chapter 3.6, we define $\to$ and $\not\to$ to be $\overset{f+1}{\longmapsto}$ and $\overset{f+1}{\not\longmapsto}$, respectively. Our necessity proof is based on the state-machine approach [32, 27]. The technique is also similar to the *withholding* mechanism, which was developed by Schmid, Weiss, and Keidar [67] to prove impossibility results and lower bound on the number of nodes for synchronous consensus under *transient link* failures in *fully-connected* graphs; however, we do not assume the transient fault model as in [67], and thus, our argument is more straightforward.

Intuition of Necessity Proof We first present the intuition behind the proof. Suppose that there exists a partition L, C, R, F where L, R are non-empty and $|F| \le f$ such that $C \cup R \not\to L$, and $L \cup C \not\to R$. Assume that the nodes in F are faulty, and the nodes in sets L, C, R are fault-free. Note that fault-free nodes are not aware of the identity of the faulty nodes.

Consider the case when all the nodes in L have input m, and all the nodes in $R \cup C$ have input M, where $m \ne M$. Suppose that the nodes in F (if non-empty) behave to nodes in L as if nodes in $R \cup C \cup F$ have input m, while behaving to nodes in R as if nodes in $L \cup C \cup F$ have input M. This behavior by nodes in F is possible, since the nodes in F are all assumed to be faulty here.

Consider nodes in L. Let N_L denote the set of incoming neighbors of L in $R \cup C$. Since $R \cup C \not\to L$, $|N_L| \le f$. Therefore, nodes in L cannot distinguish between the following two scenarios: (i) all the nodes in

N_L (if non-empty) are faulty, rest of the nodes are fault-free, and all the fault-free nodes have input m, and (ii) all the nodes in F (if non-empty) are faulty, rest of the nodes are fault-free, and fault-free nodes have input either m or M. In the first scenario, for validity, the output at nodes in L must be m. Therefore, in the second scenario as well, the output at the nodes in L must be m. We can similarly show that the output at the nodes in R must be M. Thus, if Condition BCS is not satisfied, nodes in L and R can be forced to decide on distinct values, violating the agreement property.

Necessity Proof Now, we present the proof of Lemma 10, which shows that Condition BCS is necessary for exact Byzantine consensus in synchronous systems. The proof below relies on our assumption of the system model presented in Chapter 1.2, particularly, on how messages are transmitted between nodes.

Proof:

The proof is by contradiction. Suppose that a correct Byzantine consensus algorithm, say ALGO, exists in $G(\mathcal{V}, \mathcal{E})$, and there exists a partition F, L, C, R of $\mathcal{V}$ such that $C \cup R \not\rightarrow L$ and $L \cup C \not\rightarrow R$. Thus, L has at most f incoming neighbors in $R \cup C$, and R has at most f incoming neighbors in $L \cup C$. Let us define:

$$\begin{aligned} N_L &:= \text{set of incoming neighbors of } L \text{ in } R \cup C \\ N_R &:= \text{set of incoming neighbors of } R \text{ in } L \cup C \end{aligned}$$

Then,

$$|N_L| \leq f \tag{3.15}$$

$$|N_R| \leq f \tag{3.16}$$

The behavior of each node $i \in \mathcal{V}$ when using ALGO can be modeled by a state machine that characterizes the behavior of each node $i \in \mathcal{V}$.

We construct a new network called $\mathcal{N}$, as illustrated in Figure 3.3. In $\mathcal{N}$, there are three copies of each node in C, and two copies of each node in $L \cup R \cup F$. In particular, C0 represents one copy of the nodes in C, C1 represents the second copy of the nodes in C, and C2 represents the third copy of the nodes in C. Similarly, R0 and R2 represent the two copies of the nodes in R, L0 and L1 represent the two copies of the nodes in L, and F1 and F2 represent the two copies of the nodes in F. Even though the figure shows just one vertex for C1, it represents all the nodes in C (each node in C has a counterpart in the nodes represented by C1). Same correspondence holds for other vertices in Figure 3.3.

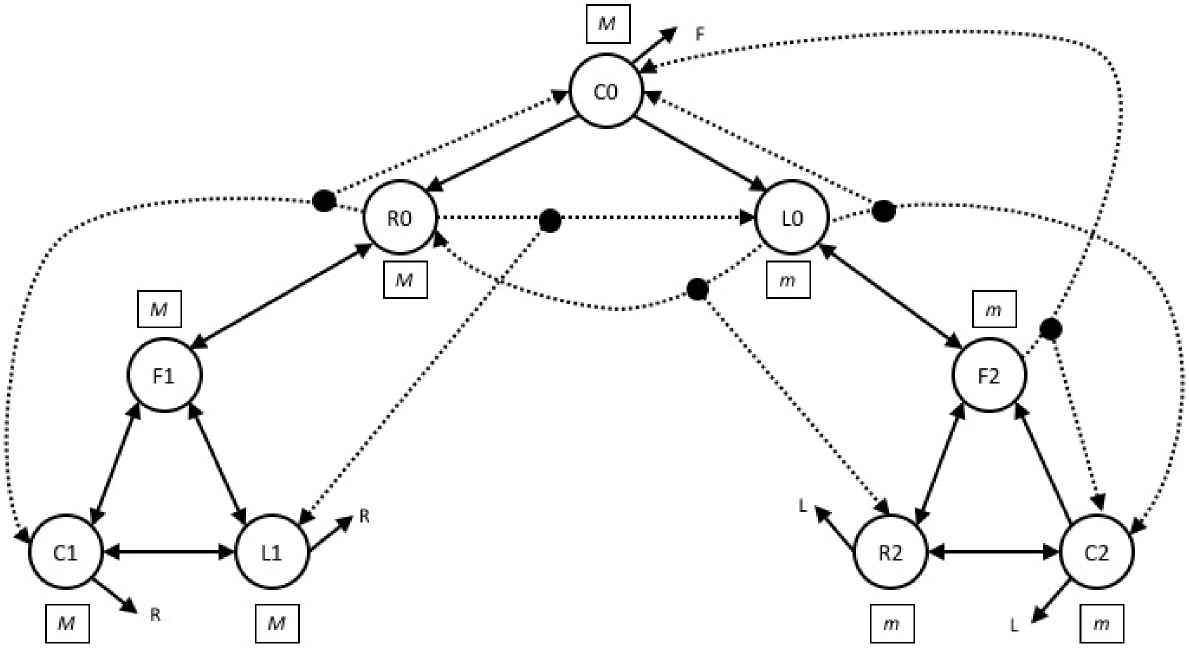

Figure 3.3: Network $\mathcal{N}$

The communication links in $\mathcal{N}$ are derived using the communication graph $G(\mathcal{V}, \mathcal{E})$. The figure shows solid edges and dotted edges, and also edges that do not terminate on one end. We describe all three types of edges below.

- *Solid edges:* If a node i has a link to node j in $G(\mathcal{V}, \mathcal{E})$, i.e., $(i, j) \in \mathcal{E}$, then each copy of node j in $\mathcal{N}$ will have a link from **one of the copies** of node i in $\mathcal{N}$. Exactly which copy of node i has link to a copy of node j is represented with the edges shown in Figure 3.3. For instance, the directed edge from vertex R0 to vertex F1 in Figure 3.3 indicates that, **if** for $r \in R$ and $k \in F$, link $(r, k) \in \mathcal{E}$, then there is a link in $\mathcal{N}$ from the copy of r in R0 to the copy of k in F1. Similarly, the directed edge from vertex F2 to vertex L0 in Figure 3.3 indicates that, **if** for $k \in F$ and $l \in L$, link $(k, l) \in \mathcal{E}$, then there is a link from the copy of k in F2 to the copy of l in L0. Other solid edges in Figure 3.3 represent other communication links in $\mathcal{N}$ similarly.

- *Dotted edges:* Dotted edges are defined similar to the solid edges, with the difference being that the dotted edges emulate a broadcast operation. Specifically, in certain cases, if link $(i, j) \in \mathcal{E}$, then one copy of node i in $\mathcal{N}$ may have links to **two** copies of node j in $\mathcal{N}$, with both copies of node j receiving identical messages from the same copy of node i. This should be viewed as a "broadcast" operation that is being emulated *unbeknownst* to the nodes in $\mathcal{N}$. There are four such "broadcast edges" in the figure, shown as dotted edges. The broadcast edge from L0 to R0 and R2 indicates that **if** for $l \in L$ and $r \in R$, link $(l, r) \in \mathcal{E}$, then messages from the copy of node l in L0 are broadcast to the copies of node r in R0 and R2 both. Similarly, the broadcast edge from R0 to C0 and C1 indicates that **if** for $r \in R$ and $c \in C$, link $(r, c) \in \mathcal{E}$, then messages from the copy of node r in R0 are broadcast to the copies of node c in C0 and C1 both. There is also a broadcast edge from L0 to C0 and C2, and another broadcast edge from R0 to L0 and L1.

- *"Hanging" edges:* Five of the edges in Figure 3.3 do not terminate at any vertex. One such edge originates at each of the vertices C1, L1, R2, C2, and C0, and each such edge is labeled as R, L or F, as explained next. A *hanging* edge signifies that the corresponding transmissions are discarded silently without the knowledge of the sender. In particular, the *hanging* edge originating at L1 with label R indicates the following: if for $l \in L$ and $r \in R$, $(r, l) \in \mathcal{E}$, then transmissions by the copy of node l in L1 to node r are silently discarded *without the knowledge* of the copy of node l in L1. Similarly, the *hanging* edge originating at C0 with label F indicates the following: if for $c \in C$ and $k \in F$, $(c, k) \in \mathcal{E}$, then transmissions by the copy of node c in C0 to node k are silently discarded *without the knowledge* of the copy of node c in C0.

It is possible to avoid using such "hanging" edges by introducing additional vertices in $\mathcal{N}$, i.e., dummy vertices that do not send any outgoing message. We choose the above approach to make the representation more compact.

Whenever $(i, j) \in \mathcal{E}$, in network $\mathcal{N}$, each copy of node j has an incoming edge from **one copy of** node i, as discussed above. The broadcast and hanging edges defined above are consistent with our *communication model* in Chapter 1.2. As noted there, each node, when sending a message, simply puts the message in the send buffer. Thus, it is possible for us to emulate hanging edges by discarding messages from the buffer, or broadcast edges by replicating the messages into two send buffers. (Nodes do not read messages in send buffers.)

Now, let us assign input of m or M, where $m \neq M$, to each of the nodes in $\mathcal{N}$. The inputs are shown next to the vertices in small rectangles Figure 3.3. For instance, M next to vertex C1 means that each node represented by C1 has input M (recall that C1 represents one copy of each node in C). Similarly, m next to vertex L0 means that each node represented by L0 has input m.

Let β denote a particular execution of ALGO in $\mathcal{N}$ given the input specified above. Now, we identify three executions of ALGO in $G(\mathcal{V}, \mathcal{E})$ with a different set of nodes of size $\leq f$ behaving faulty. The behavior of the nodes is modeled by the corresponding nodes in $\mathcal{N}$.

- **Execution α_1:**

  Consider an execution α_1 of ALGO in $G(\mathcal{V}, \mathcal{E})$, where the incoming neighbors of nodes in R that are in L or C, i.e., nodes in N_R, are faulty, with the rest of the nodes being fault-free. In addition, all the fault-free nodes have inputs M. Now, we describe the behavior of each node.

  - The behavior of fault-free nodes in R, F, $C - N_R$ and $L - N_R$ is modeled by the corresponding nodes in R0, F1, C1, and L1 in $\mathcal{N}$. For example, nodes in F send to their outgoing neighbors in L the messages sent in β by corresponding nodes in F1 to their outgoing neighbors in L1.
  - The behavior of the faulty nodes (i.e., nodes in N_R) is modeled by the behavior of the senders for the incoming links at the nodes in R0. In other words, faulty nodes are sending to their outgoing neighbors in R the messages sent in β by corresponding nodes in C0 or L0 to their outgoing neighbors in R0.

  Recall from (3.16) that $|N_R| \leq f$. Since ALGO is correct in $G(\mathcal{V}, \mathcal{E})$, the nodes in R must agree on M, because all the fault-free nodes in have input M.

- **Execution α_2:**

  Consider an execution α_2 of ALGO in $G(\mathcal{V}, \mathcal{E})$, where the incoming neighbors of nodes in L that are in R or C, i.e., nodes in N_L, are faulty, with the rest of the nodes being fault-free. In addition, all the fault-free nodes have inputs m. Now, we describe the behavior of each node.

  – The behavior of fault-free nodes in $R - N_L$, F, $C - N_L$, and L is modeled by the corresponding nodes in R2, F2, C2, and L0 in $\mathcal{N}$. For example, nodes in F send to their outgoing neighbors in R the messages sent in β by corresponding nodes in F2 to their outgoing neighbors in R2.

  – The behavior of the faulty nodes (i.e., nodes in N_L) is modeled by the behavior of the senders for the incoming links at the nodes in L0. In other words, faulty nodes are sending to their outgoing neighbors in L the messages sent in β by corresponding nodes in C0 or R0 to their outgoing neighbors in L0.

  Recall from (3.15) that $|N_L| \leq f$. Since ALGO is correct in $G(\mathcal{V}, \mathcal{E})$, the nodes in L must agree on m, because all the fault-free nodes in have input m.

- **Execution α_3:**

  Consider an execution α_3 of ALGO in $G(\mathcal{V}, \mathcal{E})$, where the nodes in F are faulty, with the rest of the nodes being fault-free. In addition, nodes in $R \cup C$ have inputs M, and the nodes in L have inputs m. Now, we describe the behavior of each node.

  – The behavior of fault-free nodes in R, C and L is modeled by the corresponding nodes in R0, C0, and L0 in $\mathcal{N}$. For example, nodes in R are sending to their outgoing neighbors in L the messages sent in β by corresponding nodes in R0 to their outgoing neighbors in L0.

  – The behavior of the faulty nodes (i.e., nodes in F) is modeled by the nodes in F1 and F2. In particular, faulty nodes in F send to their outgoing neighbors in R the messages sent in β by corresponding nodes in F1 to their outgoing neighbors in R0. Similarly the faulty nodes in F send to their outgoing neighbors in L the messages sent in β by corresponding nodes in F2 to their outgoing neighbors in L0.

  Then we make the following two observations regarding α_3:

  – Nodes in R must decide M in α_3 because, by construction, nodes in R cannot distinguish between α_1 and α_3. Recall that nodes in R decide on M in α_1.

- Nodes in L must decide m. because by construction, nodes in L cannot distinguish between α_2 and α_3. Recall that nodes in L decide on m in α_2.

Thus, in α_3, the fault-free nodes in R and L decide on different values, even though $|F| \leq f$. This violates the agreement condition, contradicting the assumption that ALGO is correct in $G(\mathcal{V}, \mathcal{E})$. Thus, the proof of Lemma 10.

□

Useful Observations Here, we prove some important properties on the graph $G(\mathcal{V}, \mathcal{E})$ implied by Condition BCS. Recall that $|\mathcal{V}| = n$.

Lemma 11 *If $G(\mathcal{V}, \mathcal{E})$ satisfies Condition BCS, then $n \geq 3f + 1$.*

Proof: For $f = 0$, the lemma is trivially true. Now consider $f > 0$. The proof is by contradiction. Suppose that $n \leq 3f$. As stated in Chapter 1.2, we assume $n \geq 2$, since consensus for $n = 1$ is trivial. Partition $\mathcal{V}$ into three subsets A, B, F such that $|F| \leq f$, $0 < |A| \leq f$, and $0 < |B| \leq f$. Such a partition can be found because $2 \leq |\mathcal{V}| \leq 3f$. Since A, B are both non-empty, and contain at most f nodes each, we have $A \overset{\mathcal{V}-F}{\not\rightsquigarrow} B$ and $B \overset{\mathcal{V}-F}{\not\rightsquigarrow} A$, violating Condition BCS. This proves the lemma. □

Corollary 1 *Suppose that a correct Byzantine consensus algorithm exists for $G(\mathcal{V}, \mathcal{E})$. Then, (i) $n \geq 3f+1$, and (ii) if $f > 0$, then each node must have at least $2f + 1$ incoming neighbors.*

Proof: Since $n \geq 3f + 1$ is a necessary condition for Byzantine consensus in undirected graphs [32, 5], it follows that $n \geq 3f + 1$ is also necessary for directed graphs. Obviously, Theorem 3 and Lemma 11 together also imply that the condition is necessary.

Now, for $f > 0$, we show that it is necessary for each node to have at least $2f + 1$ incoming neighbors. The proof is by contradiction. Suppose that for some node $i \in \mathcal{V}$, the number of incoming neighbors is at most $2f$. Partition $\mathcal{V} - \{i\}$ into two sets L and F such that L is non-empty and contains at most f incoming neighbors of i, and $|F| \leq f$. It should be easy to see that such L, F can be found, since node i has at most $2f$ incoming neighbors.

Define $A = \{i\}$ and $B = \mathcal{V} - A - F = L$. Thus, A, B, F form a partition of $\mathcal{V}$. Then, since $f > 0$ and $|A| = 1, |B| = |L| > 0$, it follows that $A \overset{\mathcal{V}-F}{\not\rightsquigarrow} B$. Also, since B contains at most f incoming neighbors of node i, and set A contains only node i, there are at most f node-disjoint (B, i)-paths. Thus, $B \overset{\mathcal{V}-F}{\not\rightsquigarrow} A$. The above two conditions violate Condition BCS. □

3.6.3 Equivalent Condition

To facilitate the discussion of exact Byzantine consensus algorithms in directed graphs, we present a condition that is equivalent to Condition BCS, namely *Condition BCS-Prop*. Recall that $\rightsquigarrow$ is introduced in Definition 2.

Definition 5 ***(Condition BCS-Prop)*** *For any partition of A, B, F of $\mathcal{V}$ such that A, B are non-empty and $|F| \leq f$, either $A \overset{\mathcal{V}-F}{\rightsquigarrow} B$ or $B \overset{\mathcal{V}-F}{\rightsquigarrow} A$.*

Lemma 12 *Condition BCS is equivalent to Condition BCS-Prop.*

Here, we first prove that *Condition BCS-Prop* implies *Condition BCS*, and then prove that *Condition BCS* implies *Condition BCS-Prop*. Thus, the two conditions are proved to be equivalent.

Two Useful Lemmas We begin with the two lemmas below (Lemmas 13 and 14). The proofs use the following version of Menger's theorem [92]. Given a graph $G(\mathcal{V}, \mathcal{E})$ and two nodes $x, y \in \mathcal{V}$, then a set $S \subseteq \mathcal{V} - \{x, y\}$ is an (x, y)-cut if there is no (x, y)-path excluding S, i.e., every path from x to y must contain some nodes in S. Define $\kappa_G(x, y) = \min\{|S| \text{ such that } S \text{ is an } (x, y)\text{-cut}\}$, and $\lambda_G(x, y) = \max\{|P| \text{ such that } P \text{ is a set of disjoint } (x, y)\text{-paths}\}$.

Theorem 4 (Menger's Theorem) *Given $G(\mathcal{V}, \mathcal{E})$ and two distinct nodes $x, y \in \mathcal{V}$ such that $(x, y) \notin \mathcal{E}$, then $\kappa_G(x, y) = \lambda_G(x, y)$.*

Lemma 13 *Assume that* Condition BCS *holds for $G(\mathcal{V}, \mathcal{E})$. For any partition A, B, F of $\mathcal{V}$, where A is non-empty, and $|F| \leq f$, if $B \not\rightarrow A$, then $A \overset{\mathcal{V}-F}{\rightsquigarrow} B$.*

Proof: Suppose that A, B, F is a partition of $\mathcal{V}$, where A is non-empty, $|F| \leq f$, and $B \not\rightarrow A$. If $B = \emptyset$, then by Definition 2, the lemma is trivially true. In the rest of this proof, assume that $B \neq \emptyset$.

Add a new (virtual) node v to graph G, such that, (i) v has no incoming edges, (ii) v has an outgoing edge to each node in A, and (iii) v has no outgoing edges to any node that is not in A. Let G_{+v} denote the graph resulting after the addition of v to $G(\mathcal{V}, \mathcal{E})$ as described above.

We want to prove that $A \overset{\mathcal{V}-F}{\rightsquigarrow} B$. Equivalently,[2] we want to prove that, in graph G_{+v}, for each $b \in B$, there exist $f + 1$ disjoint (v, b)-paths excluding F. We will prove this claim by contradiction.

[2] `Footnote:` *Justification*: Suppose that $A \overset{\mathcal{V}-F}{\rightsquigarrow} B$. By the definition of $A \overset{\mathcal{V}-F}{\rightsquigarrow} B$, for each $b \in B$, there exist at least $f + 1$ disjoint (A, b)-paths excluding F; these paths only share node b. Since v has outgoing links to all the nodes in A, this implies that there exist $f + 1$ disjoint (v, b)-paths excluding F in G_{+v}; these paths only share nodes v and b. Now, let us prove the converse. Suppose that there exist $f + 1$ disjoint (v, b)-paths excluding F in G_{+v}. Node v has outgoing links only to the nodes in A, therefore, from the $(f + 1)$ disjoint (v, b)-paths excluding F, if we delete node v and its outgoing links, then the shortened paths are disjoint (A, b)-paths excluding F.

Suppose that $A \overset{\mathcal{V}-F}{\not\rightsquigarrow} B$, and therefore, there exists a node $b \in B$ such that there are at most f disjoint (v, b) paths excluding F in G_{+v}. By construction, there is no direct edge from v to b. Then Menger's theorem [92] implies that there exists a set $F_1 \subseteq (A \cup B) - \{b\}$ with $|F_1| \leq f$, such that, in graph G_{+v}, there is no (v, b)-path excluding $F \cup F_1$. In other words, all (v, b)-paths excluding F contain at least one node in F_1.

Let us define the following sets L, R, C. Some of the sets defined in this proof are illustrated in Figure 3.4.

- $L = A$.

  L is non-empty, because A is non-empty.

- $R = \{\, i \mid i \in B - F_1$ and there exists an (i, b)-path excluding $F \cup F_1\}$.

  Thus, $R \subseteq B - F_1 \subseteq B$.

  Note that $b \in R$. Thus, R is non-empty.

- $C = B - R$.

  Thus, $C \subseteq B$. Since $R \subseteq B$, it follows that $R \cup C = B$.

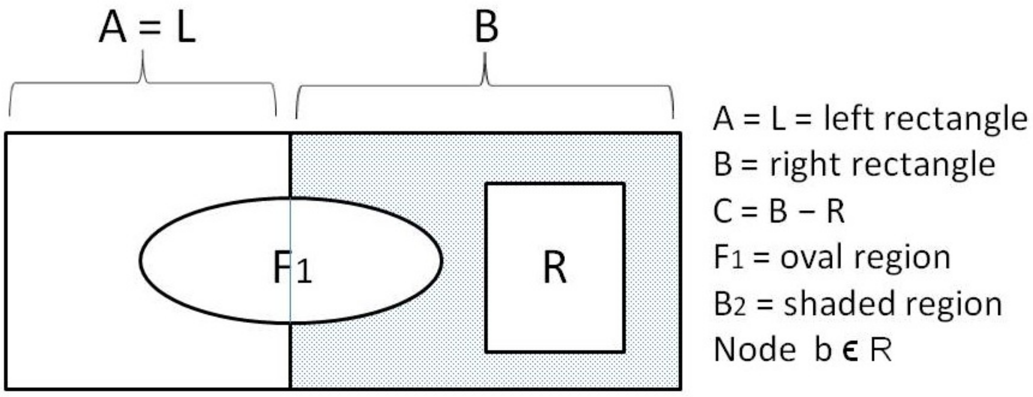

Figure 3.4: Illustration for the proof of Lemma 13

Observe that L, R, C are disjoint sets, because A and B are disjoint, and $L \cup R \cup C = A \cup B$. Since set $F_1 \subseteq A \cup B$, $L = A$, and $R \cap F_1 = \emptyset$, we have $F_1 \subseteq L \cup C$, and $F_1 \cap B \subseteq C$. Thus, set C can be partitioned into disjoint sets B_1 and B_2 such that

- $B_1 = C \cap F_1 = B \cap F_1 \subseteq C \subseteq B$, and

- $B_2 = C - B_1 \subseteq C \subseteq B$. Note that $B_2 \cap F_1 = \emptyset$.

We make the following observations:

- For any $x \in A - F_1 = L - F_1$ and $y \in R$, $(x, y) \notin \mathcal{E}$.

  Justification: Recall that virtual node v has a directed edge to x. If edge (x, y) were to exist then there would be a (v, b)-path via nodes x and y excluding $F \cup F_1$ (recall from definition of R that y has a path to b excluding $F \cup F_1$). This contradicts the definition of set F_1.

- For any $p \in B_2$, and $q \in R$, $(p, q) \notin \mathcal{E}$.

  Justification: If edge (p, q) were to exist, then there would be a (p, b)-path via node q excluding $F \cup F_1$, since q has a (q, b)-path excluding $F \cup F_1$. Then node p should have been in R by the definition of R. This is a contradiction to the assumption that $p \in B_2$, since $B_2 \cap R \subseteq C \cap R = \emptyset$.

Thus, all the incoming neighbors of set R are contained in $F \cup F_1$ (note that $F_1 = (A \cap F_1) \cup B_1$). Recall that $F_1 \subseteq L \cup C$. Since $|F_1| \leq f$, it follows that

$$L \cup C \not\rightarrow R \tag{3.17}$$

Recall that $B \not\rightarrow A$. By definitions of L, R, C above, we have $A = L$ and $B = C \cup R$. Thus,

$$C \cup R \not\rightarrow L \tag{3.18}$$

(3.17) and (3.18) together violate Condition BCS. Thus, we have proved that $A \overset{\mathcal{V}-F}{\rightsquigarrow} B$. □

Lemma 14 *Assume that Condition BCS holds for $G(\mathcal{V}, \mathcal{E})$. Consider a partition A, B, F of $\mathcal{V}$, where A, B are both non-empty, and $|F| \leq f$. If $B \overset{\mathcal{V}-F}{\not\rightsquigarrow} A$ then there exist A' and B' such that*

- *A' and B' are both non-empty,*

- *A' and B' form a partition of $A \cup B$,*

- *$A' \subseteq A$ and $B \subseteq B'$, and*

- *$B' \not\rightarrow A'$.*

Proof: Suppose that $B \overset{\mathcal{V}-F}{\not\rightsquigarrow} A$.

Add a new (virtual) node w to graph G, such that, (i) w has no incoming edges, (ii) w has an outgoing edge to each node in B, and (iii) w has no outgoing edges to any node that is not in B. Let G_{+w} denote the graph resulting after addition of w to $G(\mathcal{V}, \mathcal{E})$ as described above.

Since $B \overset{\mathcal{V}-F}{\not\rightsquigarrow} A$, for some node $a \in A$ there exist at most f disjoint (B, a)-paths excluding F. Therefore, there exist at most f disjoint (w, a)-paths excluding F in G_{+w}.[3] Also by construction, $(w, a) \notin \mathcal{E}$. Then, by Menger's theorem [92], there must exist $F_1 \subseteq (A \cup B) - \{a\}$, $|F_1| \leq f$, such that, in graph G_{+w}, all (w, a)-paths excluding F contain at least one node in F_1.

Define the following sets (also recall that $\mathcal{V} - F = A \cup B$):

- $X = \{ \, i \mid i \in \mathcal{V} - F - F_1$ and there exists an (i, a)-path excluding $F \cup F_1 \, \}$. Note that by definition, $a \in L$. Thus, L is non-empty.

- $Y = \{ \, j \mid j \in \mathcal{V} - F - F_1$ and there exists in G_{+w} a (w, j)-path excluding $F \cup F_1 \, \}$.

  Set R contains $B - F_1$ since all nodes in B have edges from w. Note that Y may be empty.

- $Z = \mathcal{V} - F - X - Y = (A \cup B) - X - Y$.

  Observe that $F_1 \subseteq C$ (because nodes of F_1 are not in $X \cup Y$). Also, by definition of Z, $Z \cap (X \cup Y) = \emptyset$.

Observe the following:

- $X \cap Y = \emptyset$, and set $X \subseteq A - F_1 \subseteq A$. Also, $A \cup B = X \cup Y \cup Z$.

  Justification: $F_1 \cap X = F_1 \cap Y = \emptyset$. By definition of F_1, all (w, a)-paths excluding F contain at least one node in F_1. If $X \cap Y$ were to be non-empty, we can find a (w, a)-path excluding $F \cup F_1$, which is a contradiction.

  Note that $\mathcal{V} - F - F_1 = (A \cup B) - F_1$; therefore, $X \subseteq (A \cup B) - F_1$. $B - F_1 \subseteq Y$, since all nodes in $B - F_1$ have links from w. Since $X \cap Y = \emptyset$, it follows that $(B - F_1) \cap X = \emptyset$, and therefore, $(A - F_1) \cap X = X$; that is, $X \subseteq A - F_1 \subseteq A$.

- For any $z \in Z - F_1$ and $x \in X$, $(z, x) \notin \mathcal{E}$.

  Justification: If such a link were to exist, then z should be in X, which is a contradiction (since $Z \cap X = \emptyset$).

- There are no links from nodes in Y to nodes in X.

  Justification: If such a link were to exist, it would contradict the definition of F_1, since we can now find a (w, a)-path excluding $F \cup F_1$.

[3] See footnote 2.

Thus, all the incoming neighbors of set X must be contained in $F \cup F_1$. Recall that $F_1 \subseteq Z$ and $|F_1| \leq f$. Thus,

$$Y \cup Z \not\to X \tag{3.19}$$

Note that $Y \cup Z$ is non-empty, since $B \cup F_1 \subseteq Y \cup Z$ and B is non-empty. Now define $A' := X$, $B' := Y \cup Z$. Observe the following:

- A' and B' form a partition of $A \cup B$.

  Justification: $X \cap (Y \cup Z) = \emptyset$ by construction; therefore $A' = X$ and $B' = Y \cup Z$ are disjoint. By the definition of sets X, Y, Z, it follows that $A' \cup B' = X \cup (Y \cup Z) = \mathcal{V} - F = A \cup B$.

- A' is non-empty and $A' \subseteq A$.

  Justification: By definition, set X contains node a. Thus, $A' = X$ is non-empty. We have already argued that $X \subseteq A$. Thus, $A' \subseteq A$.

- B' is non-empty and $B \subseteq B'$.

  Justification: Recall that $X \cap (Y \cup Z) = \emptyset$, and $X \cup Y \cup Z = A \cup B$ by definition. Thus, $Y \cup Z = (A \cup B) - X$. Since $X \subseteq A$, it follows that $B \subseteq Y \cup Z = B'$. Also, since B is non-empty, B' is also non-empty.

- $B' \not\to A'$

  Justification: Follows directly from (3.19), and the definition of A' and B'.

This concludes the proof of Lemma 14. □

Proof of Lemma 12 Now, we are ready to prove Lemma 12, i.e., showing that Condition BCS is equivalent to Condition BCS-Prop.

Lemma 15 *Condition BCS implies Condition BCS-Prop.*

Proof:

Assume that Condition BCS holds for $G(\mathcal{V}, \mathcal{E})$. Consider a partition of A, B, F of $\mathcal{V}$ such that A, B are non-empty and $|F| \leq f$. Then, we must show that either $A \overset{\mathcal{V}-F}{\rightsquigarrow} B$ or $B \overset{\mathcal{V}-F}{\rightsquigarrow} A$.

Consider two possibilities:

- $B \overset{\mathcal{V}-F}{\rightsquigarrow} A$: In this case, the proof is complete.
- $B \overset{\mathcal{V}-F}{\not\rightsquigarrow} A$: Then by Lemma 14, there exist non-empty sets A', B' that form a partition of $A \cup B$ such that $A' \subseteq A$, $B \subseteq B'$, and $B' \not\rightarrow A'$. Lemma 13 then implies that $A' \overset{\mathcal{V}-F}{\rightsquigarrow} B'$.

  Because $A' \overset{\mathcal{V}-F}{\rightsquigarrow} B'$, for each $b \in B'$, there exist $f+1$ disjoint (A', b)-paths excluding F. Since $B \subseteq B'$, it then follows that, for each $b \in B \subseteq B'$, there exist $f+1$ disjoint (A', b)-paths excluding F. Since $A' \subseteq A$, and $F \cap A = \emptyset$, each (A', b)-path excluding F is also a (A, b)-path excluding F. Thus, for each $b \in B$, there exist $f+1$ disjoint (A, b)-paths excluding F. Therefore, $A \overset{\mathcal{V}-F}{\rightsquigarrow} B$.

□

Then, we prove the other direction of Lemma 12.

Lemma 16 *Condition BCS-Prop implies Condition BCS.*

Proof: We will prove the lemma by showing that if Condition BCS is violated, then Condition BCS-Prop is violated as well.

Suppose that Condition BCS is violated. Then there exists a partition L, C, R, F of $\mathcal{V}$ such that L, R are both non-empty, $|F| \leq f$, $L \cup C \not\rightarrow R$ and $R \cup C \not\rightarrow L$.

Since $L \cup C \not\rightarrow R$, for any node $r \in R$, there exists a set F_r, $|F_r| \leq f$, such that all the $(L \cup C, r)$-paths excluding F contain at least one node in F_r. Since $L \subseteq L \cup C$, Menger's theorem [92] implies that there are at most f disjoint (L, r)-paths excluding F. Thus, because $r \in R \cup C$, $L \overset{\mathcal{V}-F}{\not\rightsquigarrow} R \cup C$.

Similarly, since $R \cup C \not\rightarrow L$, for any node $l \in L$, there exists a set F_l, $|F_l| \leq f$, such that all the $(R \cup C, l)$-paths excluding F contain at least one node in F_l. Menger's theorem [92] then implies that there are at most f disjoint $(R \cup C, l)$-paths excluding F. Thus, $R \cup C \overset{\mathcal{V}-F}{\not\rightsquigarrow} L$.

Define $A = L$, and $B = R \cup C$. Thus, A, B, F is a partition of $\mathcal{V}$ such that $|F| \leq f$ and A, B are non-empty. The two conditions derived above imply that $A \overset{\mathcal{V}-F}{\not\rightsquigarrow} B$ and $B \overset{\mathcal{V}-F}{\not\rightsquigarrow} A$, violating Condition BCS-Prop. □

Lemma 15 together with Lemma 16 prove Lemma 12, i.e., Condition BCS is equivalent to Condition BCS-Prop. Note that by Theorem 1, Condition BCS-Prop is also necessary.

3.6.4 Useful Definitions

Before proving the sufficiency of Condition BCS, we introduce some definitions and results to facilitate the discussion. In particular, we present the notion of source component and reduced graph that are useful for designing the consensus algorithms in directed graphs.

Definition 6 (Graph Decomposition) *Let H be a subgraph of $G(\mathcal{V}, \mathcal{E})$. Partition graph H into non-empty strongly connected components, $H_1, H_2, \cdots, H_h$, where h is a non-zero integer dependent on graph H, such that nodes $i, j \in H_k$ if and only if there exist (i, j)- and (j, i)-paths both excluding nodes outside H_k.*

Construct a graph H^d wherein each strongly connected component H_k above is represented by vertex c_k, and there is an edge from vertex c_k to vertex c_l if and only if the nodes in H_k have directed paths in H to the nodes in H_l.

It is known that the decomposition graph H^d is a directed *acyclic* graph [24].

Definition 7 (Source Component) *Let H be a directed graph, and let H^d be its decomposition as per Definition 6. Strongly connected component H_k of H is said to be a* source component *if the corresponding vertex c_k in H^d is not reachable from any other vertex in H^d.*

Definition 8 (Reduced Graph) *For a given graph $G(\mathcal{V}, \mathcal{E})$, and sets $F \subset \mathcal{V}$, $F_1 \subset \mathcal{V} - F$, such that $|F| \leq f$ and $|F_1| \leq f$, reduced graph $G_{F,F_1}(\mathcal{V}_{F,F_1}, \mathcal{E}_{F,F_1})$ is defined as follows: (i) $\mathcal{V}_{F,F_1} = \mathcal{V} - F$, and (ii) $\mathcal{E}_{F,F_1}$ is obtained by removing from $\mathcal{E}$ all the links incident on the nodes in F, and all the outgoing links from nodes in F_1. That is, $\mathcal{E}_{F,F_1} = \mathcal{E} - \{(i, j) \mid i \in F \;\; or \;\; j \in F\} - \{(i, j) \mid i \in F_1\}$.*

Next, we present two lemmas that show the properties of the reduced graph of G given that G satisfies Condition BCS.

Lemma 17 *Suppose that graph $G(\mathcal{V}, \mathcal{E})$ satisfies Condition BCS. For any $F \subset \mathcal{V}$ and $F_1 \subset \mathcal{V} - F$, such that $|F| \leq f$ and $|F_1| \leq f$, let S denote the set of nodes in the source component of G_{F,F_1}. Then, $S \overset{\mathcal{V}-F}{\rightsquigarrow} \mathcal{V} - F - S$.*

Proof: Since G_{F,F_1} contains non-zero number of nodes, its source component S must be non-empty. If $\mathcal{V} - F - S$ is empty, then the corollary follows trivially by Definition 2. Suppose that $\mathcal{V} - F - S$ is non-empty. Since S is a source component in G_{F,F_1}, it has no incoming neighbors in G_{F,F_1}; therefore, all of the incoming neighbors of S in $\mathcal{V} - F$ in graph $G(\mathcal{V}, \mathcal{E})$ must belong to F_1. Since $|F_1| \leq f$, we have,

$$(\mathcal{V} - S - F) \not\rightarrow S$$

Lemma 13 then implies that

$$S \overset{\mathcal{V}-F}{\rightsquigarrow} \mathcal{V} - F - S$$

□

Lemma 18 *For any $F \subset \mathcal{V}$, $F_1 \subset \mathcal{V} - F$, such that $|F| \leq f$, $|F_1| \leq f$:*

- *The source component of G_{F,F_1} is strongly connected in G_{-F}. (G_{-F} is defined in Definition 3 in Chapter 3.6.5.)*
- *The source component of G_{F,F_1} does not contain any nodes in F_1.*
- *The source component of G_{F,F_1} is unique.*

Proof: By Definition 6, each pair of nodes i, j in the source component of graph G_{F,F_1} has at least one (i, j)-path and at least one (j, i)-path consisting of nodes only in G_{F,F_1}, i.e., excluding nodes in F.

Since $F_1 \subset \mathcal{V} - F$, G_{F,F_1} contains other nodes besides F_1. Although nodes of F_1 belong to graph G_{F,F_1}, the nodes in F_1 do not have any outgoing links in G_{F,F_1}. Thus, a node in F_1 cannot have paths to any other node in G_{F,F_1}. Then, due to the connetedness requirement of a source component, it follows that no nodes of F_1 can be in the source component.

Now, we prove the third claim. The proof is by contradiction. Suppose that there exist two source components S_1 and S_2 of G_{F,F_1}. Now, consider the following set:

- $L := S_1 \cup F_1$,
- $R := S_2$, and
- $C := \mathcal{V} - L - R - F$.

Observe that L, C, R, F are disjoint, and $L \cup C \cup R \cup F = \mathcal{V}$. Also, L and R are non-empty, since S_1 and S_2 are non-empty. Then, we have

- $(L \cup C - F) \not\Rightarrow R$.

  This is because by definition, the possible incoming neighbors of R in $\mathcal{V} - R - F$ are all in F_1 and $|F_1| \leq f$.

- $(R \cup C - F) \not\Rightarrow L$.

  This is because by definition, there is no incoming neighbor of L in $\mathcal{V} - L - F$.

The two conditions above derive a contradiction. Thus, the source component is unique.

□

3.6.5 Sufficiency of Condition BCS

In the discussion below, we assume that graph $G(\mathcal{V}, \mathcal{E})$ satisfies Condition BCS, even if this is not stated explicitly again. By Lemma 12, $G(\mathcal{V}, \mathcal{E})$ satisfies Condition BCS-Prop as well. The proof consists of two parts. When $f = 0$, all the nodes are fault-free, and the proof of sufficiency is trivial. We include it below for completeness. When $f > 0$, we present Algorithm BC (Byzantine Consensus) and prove its correctness in all graphs that satisfy Condition BCS. This proves that Condition BCS is sufficient.

Sufficiency of Condition BCS when $f = 0$

Lemma 19 *When $f = 0$, Condition BCS is sufficient for achieving exact Byzantine consensus in synchronous systems.*

The proof below uses the terminologies and results presented in Chapter 3.6.4. Intuitively, Condition BCS ensures that there exists a node that can reach every other node in the graph. Hence, consensus is trivial in this case, since $f = 0$, and there is no failure.

Proof: Consider the case when $f = 0$, and suppose that the graph G satisfies Condition BCS. Consider the source component S in reduced graph $G_{\emptyset,\emptyset} = G$, as per Definition 8 in Chapter 3.6.4. Note that $F = F_1 = \emptyset$ in this case, and by definition, S is non-empty. Pick a node i in the source component. By Lemma 18 in Chapter 3.6.4, S is strongly connected in G, and thus, i has a directed path to each of the nodes in S. By Lemma 17, because $F = \emptyset$, $S \overset{\mathcal{V}}{\rightsquigarrow} \mathcal{V} - S$, i.e., for each node $j \in \mathcal{V} - S$, an (S, j)-path exists. Since S is strongly connected, an (i, j)-path also exists. Then consensus can be achieved simply by node i routing its input to all the other nodes, and requiring all the nodes to adopt node i's input as the output (or decision) for the consensus. It should be easy to see that termination, validity and agreement properties are all satisfied. □

Sufficiency of Condition BCS when $f > 0$

In the rest of our discussion below, we will assume that $f > 0$. Note that by Corollary 1, $n \geq 3f + 1$, and the number of incoming neighbors of each node is at least $2f + 1$. Also recall that here, we consider only binary inputs. The algorithm for consensus with multi-valued inputs is discussed in Chapter 3.6.6. Below, we present Algorithm BC.

Structure and Intuition of Algorithm BC:

In the algorithm, each node i maintains two state variables that are explicitly used in our algorithm: v_i and t_i. Each node maintains other state as well (such as the routes to other nodes); however, we do not introduce additional notation for that for simplicity.

- Variable v_i: Initially, v_i at any node i equals the binary input at node i. During the execution of the algorithm, v_i at node i may be updated several times. Value v_i at the end of the algorithm represents node i's decision (or output) for Algorithm BC. The output at each node is either 0 or 1. At any time during the execution of the algorithm, the value v_i at node i is said to be *valid*, if it equals some fault-free node's input. Initial value v_i at a fault-free node i is valid, because it equals its own input. Lemma 20 proved later implies that v_i at a fault-free node i always remains valid throughout the execution of Algorithm BC.

- Variable t_i: Variable t_i at any node i may take a value in $\{0, 1, \perp\}$, where $\perp$ is distinguished from 0 and 1. Algorithm BC makes use of procedures `Propagate` and `Equality` that are described soon below. These procedures take t_i as input, and possibly also modify t_i. Under some circumstances, as discussed later, state variable v_i at node i is set equal to t_i, in order to update v_i.

Algorithm BC consists of two loops, an OUTER loop, and an INNER loop. The OUTER loop is performed for each subset of nodes F, $|F| \leq f$. For each iteration of the OUTER loop, many iterations of the INNER loop are performed. The nodes in F do not participate in any of these INNER loop iterations. For a chosen F, each iteration of the INNER loop is performed for a different partition of $\mathcal{V} - F$.

Since there are at most f faults, one iteration of the OUTER loop has F exactly equal to the set of faulty nodes. Denote the actual set of faulty nodes as F^*. Algorithm BC has two properties, as proved later:

- State v_i of each fault-free node i at the end of any particular INNER loop iteration equals the state of some fault-free node at the start of that INNER loop iteration. Thus, Algorithm BC ensures that the state v_i of each fault-free node i remains valid at all times.

- By the end of the OUTER loop iteration for $F = F^*$, all the fault-free nodes reach agreement.

The above two properties ensure that, when Algorithm BC terminates, the validity and agreement properties are both satisfied.

Each iteration of the INNER loop, for a given set F, considers a partition A, B of the nodes in $\mathcal{V} - F$ such that $A \overset{\mathcal{V}-F}{\rightsquigarrow} B$. Having chosen a partition A, B, intuitively speaking, the goal of the INNER loop iteration is for the nodes in set A to attempt to influence the state of the nodes in the other partition. A suitable set $S \subseteq A \cup B$ is identified and agreed a priori using the known topology information. There are two possible cases. In Case 1 in Algorithm BC, $S \subseteq A$, and nodes in S use procedure `Equality` (step (b) in the pseudo-code) to decide the value to propagate to nodes in $\mathcal{V} - F - S$ (step (c)). In Case 2, $S \subseteq A \cup B$, and nodes in S first learn the states at nodes in A using procedure `Propagate` (step (f)), and then use procedure `Equality` (step (g)) to decide the value to propagate to nodes in $\mathcal{V} - F - S$ (step (h)). These steps ensure that if $F = F^*$, and nodes in A have the same v value, then S will propagate that value, and all nodes in $\mathcal{V} - F^* - S$ (step (d) of Case 1) or in $\mathcal{V} - F^* - (A \cap S)$ (step (i) of Case 2) will set v value equal to the value propagated by S, and thus, the agreement is achieved. As proved later, in at least one INNER loop iteration with $F = F^*$, nodes in A have the same v value.

Algorithm BC: Below, we present the pseudo-code of Algorithm BC and the two procedures.

Algorithm BC

Comment: Note that Algorithm BC can be implemented distributedly if every node has prior knowledge of the topology. For the convenience of reader, the pseudo-code below is presented in a centralized fashion.

```
(OUTER LOOP)
For each $F \subset \mathcal{V}$, where $0 \leq |F| \leq f$:

    (INNER LOOP)
    For each partition $A, B$ of $\mathcal{V} - F$ such that $A, B$ are non-empty, and $A \overset{\mathcal{V}-F}{\rightsquigarrow} B$:
    STEP 1 of INNER loop:

        • Case 1: if $A \overset{\mathcal{V}-F}{\rightsquigarrow} B$ and $B \overset{\mathcal{V}-F}{\not\rightsquigarrow} A$:
          Choose a non-empty set $S \subseteq A$ such that $S \overset{\mathcal{V}-F}{\rightsquigarrow} \mathcal{V} - F - S$, and $S$ is strongly connected in $G_{-F}$
          ($G_{-F}$ is defined in Definition 3).
            (a) At each node $i \in S$ :     $t_i := v_i$
            (b) Equality($S$)
            (c) Propagate($S, \mathcal{V} - F - S$)
            (d) At each node $j \in \mathcal{V} - F - S$ :     if $t_j \neq \perp$, then $v_j := t_j$

        • Case 2: if $A \overset{\mathcal{V}-F}{\rightsquigarrow} B$ and $B \overset{\mathcal{V}-F}{\rightsquigarrow} A$:
          Choose a non-empty set $S \subseteq A \cup B$ such that $S \overset{\mathcal{V}-F}{\rightsquigarrow} \mathcal{V} - F - S$, $S$ is strongly connected in $G_{-F}$,
          and $A \overset{\mathcal{V}-F}{\rightsquigarrow} (S - A)$.
            (e) At each node $i \in A$ :     $t_i = v_i$
            (f) Propagate($A, S - A$)
            (g) Equality($S$)
            (h) Propagate($S, \mathcal{V} - F - S$)
```

(i) At each node $j \in \mathcal{V} - F - (A \cap S)$:   if $t_j \neq \perp$, then $v_j := t_j$

STEP 2 of INNER loop:

(j) Each node $k \in F$ receives v_j from each $j \in N_k$, where N_k is a set consisting of $f+1$ of k's incoming neighbors in $\mathcal{V} - F$. If all the received values are identical, then v_k is set equal to this identical value; else v_k is unchanged.

Procedure `Propagate`(P, D):

`Propagate`(P, D) assumes that $P \subseteq \mathcal{V} - F$, $D \subseteq \mathcal{V} - F$, $P \cap D = \emptyset$ and $P \overset{\mathcal{V}-F}{\rightsquigarrow} D$. Recall that set F is the set chosen in each OUTER loop as specified by Algorithm BC.

`Propagate`(P, D)

---

(1) Since $P \overset{\mathcal{V}-F}{\rightsquigarrow} D$, for each $i \in D$, there exist at least $f+1$ disjoint (P, i)-paths that exclude F. The source node of each of these paths is in P. On each of $f+1$ such disjoint paths, the source node for that path, say s, sends t_s to node i. Intermediate nodes on these paths forward received messages as necessary.

When a node does not receive an expected message, the message content is assumed to be $\perp$.

(2) When any node $i \in D$ receives $f+1$ values along the $f+1$ disjoint paths above:
if the $f+1$ values are all equal to 0, then $t_i := 0$; else if the $f+1$ values are all equal to 1, then $t_i := 1$; else $t_i := \perp$.   (Note that := denotes the assignment operator.)

For any node $j \notin D$, t_j is not modified during `Propagate`(P, D). Also, for any node $k \in \mathcal{V}$, v_k is not modified during `Propagate`(P, D).

Procedure `Equality`(D):

`Equality`(D) assumes that $D \subseteq \mathcal{V} - F$, $D \neq \emptyset$, and for each pair of nodes $i, j \in D$, an (i, j)-path excluding F exists, i.e., D is strongly connected in G_{-F} (G_{-F} is defined in Definition 3).

`Equality`(D)

---

(1) Each node $i \in D$ sends t_i to all other nodes in D along paths excluding F.

(2) Each node $j \in D$ thus receives messages from all nodes in D. Node j checks whether values received from all the nodes in D and its own t_j are all equal, and also belong to $\{0, 1\}$. If these conditions are *not* satisfied, then $t_j := \perp$; otherwise t_j is not modified.

For any node $k \notin D$, t_k is not modified in $\texttt{Equality}(D)$. Also, for any node $k \in \mathcal{V}$, v_k is not modified in $\texttt{Equality}(D)$.

INNER Loop of Algorithm BC for $f > 0$:

Recall that we assumed that $f > 0$ and the graph satisfies both Condition BCS and Condition BCS-Prop. We discuss more details of INNER Loop. For each F chosen in the OUTER loop, the INNER loop of Algorithm BC examines each partition A, B of $\mathcal{V} - F$ such that A, B are both non-empty. From Condition BCS-Prop (Definition 5), we know that either $A \overset{\mathcal{V}-F}{\leadsto} B$ or $B \overset{\mathcal{V}-F}{\leadsto} A$. Therefore, with renaming of the sets we can ensure that $A \overset{\mathcal{V}-F}{\leadsto} B$. Then, depending on the choice of A, B, F, two cases may occur: (Case 1) $A \overset{\mathcal{V}-F}{\leadsto} B$ and $B \overset{\mathcal{V}-F}{\not\leadsto} A$, and (Case 2) $A \overset{\mathcal{V}-F}{\leadsto} B$ and $B \overset{\mathcal{V}-F}{\leadsto} A$.

In Case 1 in the INNER loop of Algorithm BC, we need to find a non-empty set $S \subseteq A$ such that $S \overset{\mathcal{V}-F}{\leadsto} \mathcal{V} - F - S$, and S is strongly connected in G_{-F} (G_{-F} is defined in Definition 3). In Case 2, we need to find a non-empty set $S \subseteq A \cup B$ such that $S \overset{\mathcal{V}-F}{\leadsto} \mathcal{V} - F - S$, S is strongly connected in G_{-F}, and $A \overset{\mathcal{V}-F}{\leadsto} (S - A)$. The following claim ensures that Algorithm BC can be executed correctly in G.

Claim 1 *Suppose that $G(\mathcal{V}, \mathcal{E})$ satisfies Condition BCS. Then,*

- *The required set S exists in both Case 1 and 2 of each INNER loop.*
- *Each node in set F has enough incoming neighbors in $\mathcal{V} - F$ to perform step (j) of Algorithm BC with $f > 0$.*

Proof: The proof below uses the terminologies and results presented in Chapter 3.6.4.

Observation:

We first prove a simple observation: *Given a partition A, B, F of $\mathcal{V}$ such that B is non-empty, and $|F| \leq f$, if $A \overset{\mathcal{V}-F}{\leadsto} B$, then size of A must be at least $f + 1$.* Now, we argue that the observation holds. By definition, there must be at least $f + 1$ disjoint (A, b)-paths excluding F for each $b \in B$. Each of these $f + 1$ disjoint paths will have a distinct *source* node in A. Therefore, such $f + 1$ disjoint paths can only exist if A contains at least $f + 1$ distinct nodes.

Proof of first claim:

Consider the two cases in the INNER loop.

- Case 1: $A \overset{\mathcal{V}-F}{\leadsto} B$ and $B \overset{\mathcal{V}-F}{\not\leadsto} A$:

  Since $B \overset{\mathcal{V}-F}{\not\leadsto} A$, by Lemma 14, there exist non-empty sets A', B' that form a partition of $A \cup B = \mathcal{V} - F$

such that $A' \subseteq A$ and

$$B' \not\rightarrow A'$$

Let F_1 be the set of incoming neighbors of A' in B'. Since $B' \not\rightarrow A'$, $|F_1| \leq f$. Then A' has no incoming neighbors in G_{F,F_1}. Therefore, the source component of G_{F,F_1} must be contained within A'. (The definition of source component is in Chapter 3.6.4.) Let S denote the set of nodes in this source component. Since S is the source component, by Lemma 17,

$$S \overset{\mathcal{V}-F}{\rightsquigarrow} \mathcal{V} - S - F.$$

Since $S \subseteq A'$ and $A' \subseteq A$, $S \subseteq A$. Then, $B \subseteq (A \cup B) - S = \mathcal{V} - S - F$; therefore, $\mathcal{V} - S - F$ is non-empty. Also, since $S \overset{\mathcal{V}-F}{\rightsquigarrow} \mathcal{V} - S - F$, set S must be non-empty (by observation above). By Lemma 18 in Chapter 3.6.4, S is strongly connected in G_{-F}. Thus, set S as required in Case 1 exists.

- Case 2: $A \overset{\mathcal{V}-F}{\rightsquigarrow} B$ and $B \overset{\mathcal{V}-F}{\rightsquigarrow} A$:

  Recall that we consider the case of $f > 0$.

  By Corollary 1, since $|\mathcal{V}| = n > 3f$, $|A \cup B| = |\mathcal{V} - F| > 2f$. In this case, we pick an arbitrary non-empty set $F_1 \subset A \cup B = \mathcal{V} - F$ such that $|F_1| = f > 0$, and find the source component of G_{F,F_1}. Let the set of nodes in the source component be denoted as S. Since S is the source component, by Lemma 17 in Chapter 3.6.4,

$$S \overset{\mathcal{V}-F}{\rightsquigarrow} \mathcal{V} - F - S$$

  Also, since $A \overset{\mathcal{V}-F}{\rightsquigarrow} B$, and $(S - A) \subseteq B$, we have $A \overset{\mathcal{V}-F}{\rightsquigarrow} (S - A)$. Also, since $\mathcal{V} - S - F$ contains F_1, and F_1 is non-empty, $\mathcal{V} - S - F$ is non-empty; also, since $S \overset{\mathcal{V}-F}{\rightsquigarrow} \mathcal{V} - S - F$, set S must be non-empty (by observation above).By Lemma 18 in Chapter 3.6.4, S is strongly connected in G_{-F}. Thus, set S as required in Case 2 exists.

Proof of second claim: Consider nodes in set F. As shown in Corollary 1, when $f > 0$, each node in $\mathcal{V}$ has at least $2f + 1$ incoming neighbors. Since $|F| \leq f$, for each $k \in F$ there must exist at least $f + 2$ incoming neighbors in $\mathcal{V} - F$. Thus, the desired set N_k exists, satisfying the requirement in step (j) of Algorithm BC.

□

Correctness of Algorithm BC for $f > 0$:

Now, we are ready to prove that Algorithm BC achieves binary exact Byzantine consensus correctly. Recall that by assumption, F^* is the actual set of faulty nodes in the network ($0 \leq |F^*| \leq f$). Thus, the set of fault-free nodes is $\mathcal{V} - F^*$. When discussing a certain INNER loop iteration, we sometimes add superscripts *start* and *end* to v_i for node i to indicate whether we are referring to v_i at the start, or at the end, of that INNER loop iteration, respectively. We first show that INNER loop preserves validity.

Lemma 20 *For any given INNER loop iteration, for each fault-free node $j \in \mathcal{V} - F^*$, there exists a fault-free node $s \in \mathcal{V} - F^*$ such that $v_j^{end} = v_s^{start}$.*

Proof: To avoid cluttering the notation, for a set of nodes X, we use the phrase

a fault-free node $j \in X$

as being equivalent to

a fault-free node $j \in X - F^*$

because all the fault-free nodes in any set X must also be in $X - F^*$.

Define set Z as the set of values of v_i at all fault-free $i \in \mathcal{V}$ at the start of the INNER loop iteration under consideration, i.e., $Z = \{v_i^{start} \mid i \in \mathcal{V} - F^* \}$.

We first prove the claim in the lemma for the fault-free nodes in $\in \mathcal{V} - F$, and then for the fault-free nodes in F. Consider the following two cases in the INNER loop iteration.

- **Case 1:** $A \overset{\mathcal{V}-F}{\rightsquigarrow} B$ and $B \overset{\mathcal{V}-F}{\not\rightsquigarrow} A$:

  Observe that, in Case 1, v_i remains unchanged for all fault-free $i \in S$. Thus, $v_i^{end} = v_i^{start}$ for $i \in S$, and hence, the claim of the lemma is trivially true for these nodes. We will now prove the claim for fault-free $j \in \mathcal{V} - F - S$.

  - step (a): Consider a fault-free node $i \in S$. At the end of step (a), t_i is equal to v_i^{start}. Thus, $t_i \in Z$.
  - step (b): In step (b), step 2 of `Equality`(S) either keeps t_i unchanged at fault-free node $i \in S$ or modifies it to be $\perp$. Thus, now $t_i \in Z \cup \{\perp\}$.
  - step (c): Consider a fault-free node $j \in \mathcal{V} - F - S$. During `Propagate`$(S, \mathcal{V} - F - S)$, j receives $f + 1$ values along $f + 1$ disjoint paths originating at nodes in S. Therefore, at least one of the $f + 1$ values is received along a path that contains only fault-free nodes; suppose that the value received by node j along this fault-free path is equal to α. As observed above in step (b), t_i at all fault-free nodes $i \in S$ is in $Z \cup \{\perp\}$. Thus, $\alpha \in Z \cup \{\perp\}$. Therefore, at fault-free node $j \in \mathcal{V} - F - S$, step 2 of `Propagate`$(S, \mathcal{V} - F - S)$ will result in $t_j \in \{\alpha, \perp\} \subseteq Z \cup \{\perp\}$.
  - step (d): Then it follows that, in step (d), at fault-free $j \in \mathcal{V} - F - S$, if v_j is updated, then $v_j^{end} \in Z$. On the other hand, if v_j is not updated, then $v_j^{end} = v_j^{start} \in Z$.

- **Case 2:** $A \overset{\mathcal{V}-F}{\rightsquigarrow} B$ and $B \overset{\mathcal{V}-F}{\rightsquigarrow} A$:

  Observe that, in Case 2, v_j remains unchanged for all fault-free $j \in A \cap S$; thus $v_j^{end} = v_j^{start}$ for these nodes. Now, we prove the claim in the lemma for fault-free $j \in \mathcal{V} - F - (A \cap S)$.

  - step (e): For any fault-free node $i \in A$, at the end of step (e), $t_i \in Z$.
  - step (f): Consider a fault-free node $m \in S - A$. During $\texttt{Propagate}(A, S - A)$, m receives $f+1$ values along $f+1$ disjoint paths originating at nodes in A. Therefore, at least one of the $f+1$ values is received along a path that contains only fault-free nodes; suppose that the value received by node m along this fault-free path is equal to $\gamma \in Z$. Therefore, at node $m \in S - A$, $\texttt{Propagate}(A, S - A)$ will result in t_m being set to a value in $\{\gamma, \perp\} \subseteq Z \cup \{\perp\}$. Now, for $m \in S \cap A$, t_m is not modified in step (f), and therefore, for fault-free $m \in S \cap A$, $t_m \in Z$. Thus, we can conclude that, at the end of step (f), for all fault-free nodes $m \in S$, $t_m \in Z \cup \{\perp\}$.
  - step (g): In step (g), at each $m \in S$, $\texttt{Equality}(S)$ either keeps t_m unchanged, or modifies it to be $\perp$. Thus, at the end of step (g), for all fault-free $m \in S$, t_m remains in $Z \cup \{\perp\}$.
  - step (h): Consider a fault-free node $j \in \mathcal{V} - F - S$. During $\texttt{Propagate}(S, \mathcal{V} - F - S)$, j receives $f+1$ values along $f+1$ disjoint paths originating at nodes in S. Therefore, at least one of the $f+1$ values is received along a path that contains only fault-free nodes; suppose that the value received by node j along this fault-free path is equal to β. As observed above, after step (g), for each fault-free node $m \in S$, $t_m \in Z \cup \{\perp\}$. Therefore, $\beta \in Z \cup \{\perp\}$, and at node $j \in \mathcal{V} - F - S$, $\texttt{Propagate}(S, \mathcal{V} - F - S)$ will result in t_j being set to a value in $\{\beta, \perp\} \subseteq Z \cup \{\perp\}$.
  - step (i): From the discussion of steps (g) and (h) above, it follows that, in step (i), if v_j is updated at a fault-free $j \in \mathcal{V} - F - (S \cap A)$, then $v_j^{end} \in Z$; on the other hand, if v_j is not modified, then $v_j^{end} = v_j^{start} \in Z$.

Now, consider a fault-free node $k \in F$. Step (j) uses set $N_k \subset \mathcal{V} - F$ such that $|N_k| = f+1$. As shown above, at the start of step (j), $v_j^{end} \in Z$ at all fault-free $j \in \mathcal{V} - F$. Since $|N_k| = f+1$, at least one of the nodes in N_k is fault-free. Thus, of the $f+1$ values received by node k, at least one value must be in Z. It follows that if node k changes v_k in step (j), then the new value will also in Z; on the other hand, if node k does not change v_k, then it remains equal to $v_k^{start} \in Z$. □

Lemma 21 *Algorithm BC satisfies the validity property for Byzantine consensus.*

Proof: Recall that the state v_i of a fault-free node i is *valid* if it equals the input at a fault-free node. For each fault-free $i \in \mathcal{V}$, initially, v_i is valid. Lemma 20 implies that after each INNER loop iteration, v_i

remains valid at each fault-free node i. Thus, when Algorithm BC terminates, v_i at each fault-free node i will satisfy the *validity* property for Byzantine consensus, as stated in Chapter 3.6. □

Lemma 22 *Algorithm BC satisfies the termination property for Byzantine consensus.*

Proof: Recall that we are assuming a synchronous system, and the graph $G(\mathcal{V}, \mathcal{E})$ is finite. Thus, Algorithm BC performs a finite number of OUTER loop iterations, and a finite number of INNER loop iterations for each choice of F in the OUTER loop, the number of iterations being a function of graph $G(\mathcal{V}, \mathcal{E})$. Hence, the termination property is satisfied. □

As mentioned in the beginning of this Chapter, Algorithm BC has poor time complexity. A more efficient algorithm is left as a future work. The next lemma shows the final correctness property.

Lemma 23 *Algorithm BC satisfies the agreement property for Byzantine consensus.*

Proof: Recall that F^* denotes the actual set of faulty nodes in the network ($0 \leq |F^*| \leq f$).

Since the OUTER loop of Algorithm BC considers all possible $F \subseteq \mathcal{V}$ such that $|F| \leq f$, eventually, the OUTER loop will be performed with $F = F^*$.

In the INNER loop for $F = F^*$, different partitions A, B of $\mathcal{V} - F = \mathcal{V} - F^*$ will be considered. We will say that such a partition A, B is a "conformant" partition if $v_i = v_j$ for all $i, j \in A$, and $v_i = v_j$ for all $i, j \in B$. A partition A, B that is not conformant is said to be "non-conformant". Further, we will say that an INNER loop iteration is a "deciding" iteration if one of the following condition is true.

C1 : The A, B partition of $\mathcal{V} - F$ considered in the iteration is conformant.

In Case 1 with conformant partition, every node in S has the same value t after step (a). Hence, at the end of step (b), every node in S has the same value t. Now, consider Case 2 with conformant partition. Denote the value of all the nodes in A by α ($\alpha \in \{0, 1\}$). Then, in step (e), each node i in A (including $S \cap A$) sets t_i equal to α. In step (f), all the nodes in $S \cap B$ receive identical values α from nodes in A, and hence, they set value t equal to α. Therefore, every node in S has the same value $t \in \{0, 1\}$ at the end of step (g).

C2 : The A, B partition of $\mathcal{V} - F$ considered in the iteration is non-conformant; however, the values at the nodes are such that, at the end of step (b) of Case 1, or at the end of step (g) of Case 2 (depending on which case applies), every node in the corresponding set S has the same value $t \in \{0, 1\}$. (The definition of source component is in Chapter 3.6.4.) That is, for all $i, j \in S,\ t_i = t_j$.

In both C1 and C2, all the nodes in the corresponding source component S have the identical value t in the deciding iteration (at the end of step (b) of Case 1, and at the end of step (g) of Case 2). The iteration that is not deciding is said to be "non-deciding".

Claim 2 *In the INNER loop with $F = F^*$, value v_i for each fault-free node i will stay unchanged in every non-deciding iteration.*

Proof: Suppose that $F = F^*$, and the INNER loop iteration under consideration is a non-deciding iteration. Observe that since the paths used in procedures `Equality` and `Propagate` exclude F, none of the faulty nodes can affect the outcome of any INNER loop iteration when $F = F^*$. Thus, during `Equality`(S) (step (b) of Case 1, and step (g) of Case 2), each node in S can receive the value from other nodes in S correctly. Observe that C1 and C2 above together imply that in the deciding iteration, all nodes in S have the same value $t \in \{0, 1\}$ (after step (b) of Case 1 or step (g) of Case 2). Thus, in a non-deciding iteration, there is a pair of nodes $j, k \in S$ such that $t_j \neq t_k$ (after step (b) of Case 1 or step (g) of Case 2). Then, every node in S will set value t to be $\perp$ at the end of `Equality`(S) of the non-deciding iteration. Hence, every node in $\mathcal{V} - F - S$ will receive $f + 1$ copies of $\perp$ after `Propagate`($S, \mathcal{V} - F - S$) (step (c) of Case 1, and step (h) of Case 2), and will set value t to $\perp$. Finally, at the end of the INNER loop iteration, the value v at each node stays unchanged based on the following two observations:

- nodes in S in Case 1, and in $A \cap S$ in Case 2, will not change value v as specified by Algorithm BC, and

- $t_i = \perp$ for each node $i \in \mathcal{V} - F - S$ in Case 1, and for each node $i \in \mathcal{V} - F - (A \cap S)$ in Case 2.

Thus, no node in $\mathcal{V} - F$ will change their v value (where $F = F^*$).

Note that by assumption, there is no fault-free node in $F = F^*$, and hence, we do not need to consider STEP 2 of the INNER loop. Therefore, Claim 2 is proved. □

Let us divide the INNER loop iterations for $F = F^*$ into three phases:

- Phase 1: INNER loop iterations before the first deciding iteration with $F = F^*$.

- Phase 2: The first deciding iteration with $F = F^*$.

- Phase 3: Remaining INNER loop iterations with $F = F^*$.

Claim 3 *At least one INNER loop iteration with $F = F^*$ is a deciding iteration.*

Proof: The input at each node is in $\{0, 1\}$. Therefore, by repeated application of Lemma 20, it is always true that $v_i \in \{0, 1\}$ for each fault-free node i. Thus, when the OUTER iteration for $F = F^*$ begins, a conformant partition exists (in particular, set A containing all fault-free nodes with v value 0, and set B containing the remaining fault-free nodes, or vice-versa.) By Claim 2, nodes in $\mathcal{V} - F$ will not change values during non-deciding iterations. Then, since the INNER loop considers all partitions of $\mathcal{V} - F$, the INNER loop will eventually consider either the above conformant partition, or sometime prior to considering the above conformant partition, it will consider a non-conformant partition with properties in (C2) above. □

Thus, Phase 2 will be eventually performed when $F = F^*$. Now, let us consider each phase separately:

- Phase 1: Recall that all the nodes in $\mathcal{V} - F = \mathcal{V} - F^*$ are fault-free. By Claim 2, the v_i at each fault-free node $i \in \mathcal{V} - F$ stays unchanged.
- Phase 2: Now, consider the first deciding iteration of the INNER loop.

  Recall from Algorithm BC that a suitable set S is identified in each INNER loop iteration. We will show that in the deciding iteration, every node in S will have the same t value. Consider two scenarios:

  - The partition is non-conformant: Then by definition of deciding iteration, we can find an $\alpha \in \{0, 1\}$ such that $v_i = \alpha$ for all $i \in S$ after step (b) of Case 1, or after step (g) of Case 2.
  - The partition is conformant: Let $v_i = \alpha$ for all $i \in A$ for $\alpha \in \{0, 1\}$. Such an α exists because the partition is conformant.
    - Case 1: In this case, recall that $S \subseteq A$. Therefore, after steps (a) and (b) both, t_j at all $j \in S$ will be identical, and equal to α.
    - Case 2: This is similar to Case 1. At the end of step (e), for all nodes $i \in A$, $t_i = \alpha$. After step (f), for all nodes $i \in S \cup A$, $t_i = \alpha$. Therefore, after step (g), for all nodes $i \in S$, t_i will remain equal to α.

  Thus, in both scenarios above, we found a set S and α such that for all $i \in S$, $t_i = \alpha \ \in \{0, 1\}$ after step (b) in Case 1, and after step (g) in Case 2.

  Then, consider the remaining steps in the deciding iteration.

  - Case 1: During `Propagate`$(S, \mathcal{V} - F - S)$, each node $k \in \mathcal{V} - F - S$ will receive $f + 1$ copies of α along $f + 1$ disjoint paths, and set $t_k = \alpha$ in step (c). Therefore, each node $k \in \mathcal{V} - F - S$ will update its v_k to be α in step (d). (Each node $p \in S$ does not modify its v_p, which is already equal to α.)

- Case 2: After step (h), $t_j = \alpha$ for all $j \in (\mathcal{V} - F - S) \cup S$. Thus, each node $k \in \mathcal{V} - F - (A \cap S)$ will update v_k to be α. (Each node $p \in A \cap S$ does not modify its v_p, which is already equal to α.)

Thus, in both cases, at the end of STEP 1 of the INNER loop, for all $k \in \mathcal{V} - F = \mathcal{V} - F^*$, $v_k = \alpha$.

Since all nodes in F^* are faulty, agreement has been reached at this point. The goal now is to show that the agreement property is not violated by actions taken in any future INNER loop iterations.

- Phase 3: At the start of Phase 3, for each fault-free node $k \in \mathcal{V} - F^*$, we have $v_k = \alpha \in \{0, 1\}$. Then by Lemma 20, all future INNER loop iterations cannot assign any value other than α to any node $k \in \mathcal{V} - F^*$.

After Phase 3 with $F = F^*$, Algorithm BC may perform OUTER loop iterations for other choices of set F. However, due to Lemma 20, the value v_i at each $i \in \mathcal{V} - F^*$ (i.e., all fault-free nodes) continues being equal to α.

Thus, Algorithm BC satisfies the *agreement* property. □

Theorem 5 *Algorithm BC is correct, i.e, it satisfies the agreement, validity, and termination conditions.*

Proof: The theorem follows from Lemmas 21, 22 and 23. □

Theorem 5 proves that Condition BCS is sufficient for achieving binary exact Byzantine consensus in synchronous systems when $f > 0$.

3.6.6 Application to Multi-Valued Consensus

Algorithm BC can be used to solve exact Byzantine consensus under two (weaker) versions of *multi-valued* consensus. These versions are typical in the literature [5, 52].

- Version I: The first version of *multi-valued* consensus has the following properties:
  - **Agreement**: the output (i.e., decision) at all the fault-free nodes must be identical.
  - **Validity**: If all fault-free nodes have the same input, then the output of every fault-free node equals its input.
  - **Termination**: every fault-free node eventually decides on an output.

  Under these conditions, if all the fault-free nodes do not have the same multi-valued input, then it is possible for the fault-free nodes to agree on a value that is not an input at any fault-free node. This multi-valued consensus problem for L-bit input values can be solved by *Algorithm LBC*: executing L instance of Algorithm BC, one instance for each bit of the input, on graphs that satisfy Condition BCS. The 1-bit output of each of the L instances put together form the L-bit output of the multi-valued consensus problem. Correctness of Algorithm LBC follows from Theorem 5.

- Version II: The second version of *multi-valued* consensus has the following properties:

  - **Agreement**: the output (i.e., decision) at all the fault-free nodes must be identical.
  - **Validity I**: If all fault-free nodes have the same input, then the output of every fault-free node equals its input.
  - **Validity II**: If all fault-free nodes do not have the same input, then the output of each fault-free node is either an input of a fault-free node or a special value $\perp$.
  - **Termination**: every fault-free node eventually decides on an output.

  Version II requires an extra validity condition – validity II. This multi-valued consensus problem for L-bit input values can be solved using the following algorithm (Algorithm LBC2). Denote by x_i the input at node i.

  - Execute Algorithm LBC using input x_i. Denote by w_i the output at node i of LBC. Note that w_i is an L-bit value.
  - For each node i, set $z_i := 1$ if $w_i = x_i$; otherwise, set $z_i := 0$.
  - Execute Algorithm BC using input z_i. Denote by y_i the output at node i of BC. Note that y_i is an 1-bit value.
  - Output w_i if $y_i = 1$; otherwise, output $\perp$.

  Agreement and termination follow trivially from the correctness of Algorithm BC and Algorithm LBC. Validity I also holds because if all the fault-free nodes have the same input, then at the end of first step of Algorithm LBC2, every fault-free node i has the same w_i, which equals x_i. Thus, at the second step, every fault-free node i sets $z_i := 1$. Consequently, node i outputs $w_i = x_i$ due to the correctness of Algorithm BC. Now, we show that Algorithm LBC2 satisfies validity II. The proof is by contradiction. Suppose that fault-free node i outputs a value $w_i := X$, which does not equal $\perp$ or any input at fault-free nodes. Since node i outputs X, the output of Algorithm BC at the third step must be 1, i.e., $y_i = 1$. This together with the correctness of Algorithm BC imply that there exists some fault-free node j that has input $z_j := 1$ at the second step. Thus, $X = w_j = x_j$, an input at fault-free node j. A contradiction.

The discussion above shows that Condition BCS is sufficient for these two versions of multi-valued Byzantine consensus. However, if the above validity condition for multi-valued consensus is made stronger, to require that the output value must be the multi-valued input of a fault-free node, then it is not conclusive whether Condition BCS is sufficient or not.

3.7 Discussion

As noted in Chapter 3.3, Condition CCS, CCA, and BCS capture how information can "flow between" different subsets of fault-free nodes despite the presence of faulty nodes under different synchrony assumptions. This section compares the three conditions and the condition for undirected graphs identified in [27, 32].

3.7.1 Comparison of Condition CCS, CCA, and BCS

The following lemma discusses the relationships among the three tight conditions identified in this Chapter.

Lemma 24 *Conditions CCS, CCA and BCS are progressively stronger. In particular, (i) Condition BCS implies Condition CCA, but not vice-versa, and (ii) Condition CCA implies Condition CCS, but not vice-versa.*

Proof:

- Proof of claim (i):

  It should be easy to see that Condition CCA can be viewed as a special case of Condition BCS, if we force set F in Condition BCS to be an empty set. Therefore, Condition BCS implies Condition CCA.

  A clique consisting of $2f+1$ nodes satisfies Condition CCA but not Condition BCS; thus, (not surprisingly) Condition CCA does not imply BCS.

- Proof of claim (ii):

  We will prove that CCA implies CCS by contradiction. Suppose that graph G does not satisfy Condition CCS, i.e., there exists a node partition F, L, C, R of $\mathcal{V}$ such that $L \cup C \overset{1}{\not\mapsto} R$ and $R \cup C \overset{1}{\not\mapsto} L$. Then, define $C' = C \cup F$. Due to the fact that $|F| \leq f$, $L \cup C' \overset{f+1}{\not\mapsto} R$ and $R \cup C' \overset{f+1}{\not\mapsto} L$, violating Condition CCA. This proves that CCA implies CCS.

  Consider the example network in Figure 3.2 in Chapter 3.3. This network tolerates 1 crash fault in synchronous systems, since it satisfies Condition CCS; however, it does not satisfy Condition CCA when $L = \{v_1\}, C = \emptyset$ and $R = \{v_2, v_3\}$. Thus, CCS does not imply CCA.

□

3.7.2 Comparison of Conditions in Undirected and Directed Graphs

Previously necessary conditions for undirected graphs [32, 27, 5, 52] all imply that each pair of fault-free nodes can communicate *reliably* despite the presence of f faulty nodes. In particular, this is true due to $f+1$ connectivity in case of crash faults, and $2f+1$ connectivity for Byzantine faults. Specifically, $2f+1$-connectivity implies the presence of $2f+1$ node-disjoint paths between nodes that are not neighbors, allowing each pair of nodes to communicate reliably despite f Byzantine faulty nodes. In contrast, in *directed graphs*, to be able to achieve consensus, it is *not necessary* for **all** node pairs to be able to communicate with each other reliably (even in just one direction). This is a manifestation of the "asymmetry" noted in our discussion above. For instance, the network in Figure 3.2 can tolerate 1 crash fault in a synchronous setting; however, v_3 does not have paths to the other nodes. Now, consider a network that consists of 6 nodes, 4 of which (say, w_1, w_2, w_3, w_4) form a clique, and also have directed links to nodes w_5 and w_6. Node w_5 has no path to w_6 and vice versa. Yet, Byzantine consensus can be achieved easily by first reaching consensus within the 4-node clique, and then propagating the consensus value (for the 4-node consensus) to nodes w_5 and w_6. Nodes w_5 and w_6 can choose majority of the values received from the nodes in the 4-node clique as its own output. It should be easy to see that this algorithm works correctly for inputs in $\{0,1\}$ as required in the Byzantine consensus formulation considered in this work. However, nodes w_5 and w_6 cannot communicate reliably with each other (in either direction).

There exists a family of graphs that satisfy Condition BCS wherein reliable communication may not be feasible in either direction across a certain cut. In particular, consider the network in Figure 3.1 in Chapter 3.3 again. Observe that there are only 4 directed links from K_1 to K_2, and 4 directed links from K_2 to K_1. Thus, reliable communication is *not guaranteed* across the cut (K_1, K_2) in either direction when $f = 2$ (Byzantine faults). Yet, Byzantine consensus is achievable in synchronous systems since this graph satisfies Condition BCS for $f = 2$. We prove this claim by introducing a new family of graphs, namely 2-clique Network.

2-clique Network Here, we formally introduce 2-clique network, and prove that the any 2-clique network satisfies Condition BCS,but each pair of the nodes in the network may not be able to communicate reliably with each other.

Definition 9 *A graph $G(\mathcal{V}, \mathcal{E})$ consisting of $n = 6f+2$ nodes, where f is a positive even integer, is said to be a* 2-clique network *if all the following properties are satisfied:*

- *It includes two disjoint cliques, each consisting of $3f+1$ nodes. Suppose that the nodes in the two cliques are specified by sets K_1 and K_2, respectively, where $K_1 = \{u_1, u_2, \cdots, u_{3f+1}\} \subset \mathcal{V}$, and $K_2 = \mathcal{V} - K_1 = \{w_1, w_2, \cdots, w_{3f+1}\}$. Thus, $(u_i, u_j) \in \mathcal{E}$ and $(w_i, w_j) \in \mathcal{E}$, for $1 \leq i, j \leq 3f+1$ and $i \neq j$,*
- *$(u_i, w_i) \in \mathcal{E}$, for $1 \leq i \leq \frac{3f}{2}$ and $i = 3f+1$, and*
- *$(w_i, u_i) \in \mathcal{E}$, for $\frac{3f}{2} + 1 \leq i \leq 3f$ and $i = 3f+1$.*

Figure 3.1 is the 2-clique network for $f = 2$. Note that Chapter 3.6.5 proves that Byzantine consensus is possible in all graphs that satisfy the necessary condition. Therefore, consensus is possible in the 2-clique network as well.

We first prove the following lemma for any graph $G(\mathcal{V}, \mathcal{E})$ that satisfies the necessary condition.

Lemma 25 *Let A, B, C, F be disjoint subsets of $\mathcal{V}$ such that $|F| \leq f$ and A, B, C are non-empty. Suppose that $A \overset{\mathcal{V}-F}{\rightsquigarrow} B$ and $A \cup B \overset{\mathcal{V}-F}{\rightsquigarrow} C$. Then, $A \overset{\mathcal{V}-F}{\rightsquigarrow} B \cup C$.*

Proof: The proof is by contradiction. Suppose that

- $A \overset{\mathcal{V}-F}{\rightsquigarrow} B$,
- $A \cup B \overset{\mathcal{V}-F}{\rightsquigarrow} C$, and
- $A \overset{\mathcal{V}-F}{\not\rightsquigarrow} B \cup C$.

By Definition 2 and Menger's Theorem [92], the third condition implies that there exists a node $v \in B \cup C$ and a set of nodes $P \subseteq \mathcal{V} - F - \{v\}$ such that $|P| \leq f$, and all (A, v)-paths excluding F contain at least one node in P. In other words, there is no (A, v)-path excluding $F \cup P$. Observe that, because $A \overset{\mathcal{V}-F}{\rightsquigarrow} B$, v cannot be in B; therefore v must belong to set C.

Let us define the sets X and Y as follows:

- Node $x \in X$ if and only if $x \in \mathcal{V} - F - P$ and there exists an (A, x)-path excluding $F \cup P$. It is possible that $P \cap A \neq \emptyset$; thus, the (A, x)-path cannot contain any nodes in $P \cap A$.
- Node $y \in Y$ if and only if $y \in \mathcal{V} - F - P$ and there exists an (y, v)-path excluding $F \cup P$.

By the definition of X and Y, it follows that for any $x \in X$, $y \in Y$, there cannot be any (x, y)-path excluding $F \cup P$. Also, since $A \overset{\mathcal{V}-F}{\rightsquigarrow} B$, for each $b \in B - P$, there must exist an (A, b)-path excluding $F \cup P$; thus, $B - P \subseteq X$, and $B \subseteq X \cup P$. For each node $a \in A - P$, there exists an (A, a)-path excluding $F \cup P$, since each node has a path to itself by definition. Thus, $A \subseteq X \cup P$.

By definition of X, there are no $(X \cup P, v)$-paths excluding $F \cup P$. Thus, because $A \cup B \subseteq X \cup P$, there are no $(A \cup B, v)$-paths excluding $F \cup P$. Therefore, since $v \in C$, $A \cup B \overset{\mathcal{V}-F}{\not\rightsquigarrow} C$. This is a contradiction to the second condition above. □

Now, we use Lemma 25 to prove the following Lemma.

Lemma 26 *Suppose that $G(\mathcal{V}, \mathcal{E})$ is a 2-clique network. Then graph G satisfies Condition BCS-Prop and thus, Condition BCS.*

Proof:

Consider a partition A, B, F of $\mathcal{V}$, where A and B are both non-empty, and $|F| \leq f$. Recall from Definition 9 that K_1, K_2 also form a partition of $\mathcal{V}$.

Define $A_1 = A \cap K_1, A_2 = A \cap K_2, B_1 = B \cap K_1, B_2 = B \cap K_2, F_1 = F \cap K_1$ and $F_2 = F \cap K_2$.

Define $\mathcal{E}'$ to be the set of directed links from the nodes in K_1 to the nodes in K_2, or vice-versa. Thus, there are $\frac{3f}{2} + 1$ directed links in $\mathcal{E}'$ from the nodes in K_1 to the nodes in K_2, and the same number of links from the nodes in K_2 to the nodes in K_1. Each pair of links in $\mathcal{E}'$, with the exception of the link pair between a_{3f+1} and b_{3f+1}, is node disjoint. Since $|F| \leq f$, it should be easy to see that, at least one of the two conditions below is true:

(a) There are at least $f + 1$ directed links from the nodes in $K_1 - F$ to the nodes in $K_2 - F$.

(b) There are at least $f + 1$ directed links from the nodes in $K_2 - F$ to nodes the in $K_1 - F$.

Without loss of generality, suppose that condition (a) is true. Therefore, since $|K_1 - F| \geq 2f + 1$ and the nodes in $K_2 - F$ form a clique, it follows that $K_1 - F \overset{\mathcal{V}-F}{\rightsquigarrow} K_2 - F$. Then, because $K_1 - F = A_1 \cup B_1$ and $K_2 - F = A_2 \cup B_2$, we have

$$A_1 \cup B_1 \overset{\mathcal{V}-F}{\rightsquigarrow} A_2 \cup B_2. \tag{3.20}$$

$|K_1 - F| \geq 2f + 1$ also implies that either $|A_1| \geq f + 1$ or $|B_1| \geq f + 1$. Without loss of generality, suppose that $|A_1| \geq f + 1$. Then, since the nodes in $A_1 \cup B_1$ form a clique, it follows that $A_1 \overset{\mathcal{V}-F_1-K_2}{\rightsquigarrow} B_1$ (recall that $\mathcal{V} - F_1 - K_2 = A_1 \cup B_1$). Since $\mathcal{V} - F_1 - K_2 \subset \mathcal{V} - F$, we have

$$A_1 \overset{\mathcal{V}-F}{\rightsquigarrow} B_1 \tag{3.21}$$

(3.20) and (3.21), along with Lemma 25 above imply that $A_1 \overset{\mathcal{V}-F}{\rightsquigarrow} B_1 \cup A_2 \cup B_2$. Therefore, $A_1 \overset{\mathcal{V}-F}{\rightsquigarrow} B_1 \cup B_2$, and $A_1 \cup A_2 \overset{\mathcal{V}-F}{\rightsquigarrow} B_1 \cup B_2$. Since $A = A_1 \cup A_2$ and $B = B_1 \cup B_2$, $A \overset{\mathcal{V}-F}{\rightsquigarrow} B$. □

3.8 Summary

Necessary and sufficient conditions for solving the following problems in *directed* graphs are presented in this Chapter: (i) exact crash-tolerant consensus in synchronous systems, (ii) approximate crash-tolerant consensus in asynchronous systems, and (iii) exact Byzantine consensus in synchronous systems. Development of efficient algorithms for solving these problems is a topic for future work. Also, tight condition for approximate Byzantine asynchronous consensus in directed graphs remains unknown.

Chapter 4

Iterative Approximate Byzantine Consensus Under f-total Faults

4.1 Introduction

In this Chapter, we consider "iterative" algorithms for achieving approximate Byzantine consensus in synchronous systems (with the exception of Chapter 4.6). Similar to Chapter 3, we study the f-total fault model, where up to f nodes may become Byzantine faulty, as discussed in Chapter 1.2. Note that we consider only Byzantine node failures, and all links are assumed to be reliable in this Chapter. The *iterative approximate Byzantine consensus* (IABC) algorithms of interest have the following properties, which we will soon state more formally:

- *Initial state* of each node is equal to a real-valued *input* provided to that node.
- *Validity* condition: After each iteration of an IABC algorithm, the state of each fault-free node must remain in the *convex hull* of the states of the fault-free nodes at the end of the *previous* iteration.
- *Convergence* condition: For any $\epsilon > 0$, after a sufficiently large number of iterations, the states of the fault-free nodes are guaranteed to be within ϵ of each other.

For ease of analysis, this Chapter adopts convergence condition, which has a slightly different presentation from ϵ-agreement used in Chapter 3. However, it should be obvious that these two conditions are equivalent.

In this Chapter, for the existence of a correct IABC algorithm, we derive a necessary condition that must be satisfied by the underlying communication graph. For graphs that satisfy this necessary condition, we show the correctness of a specific IABC algorithm, proving that the necessary conditions are tight. This Chapter is a joint work with Nitin H. Vaidya and Guanfeng Liang, and the results are published in [89].

4.2 IABC Algorithms

Here, we describe the structure of the *iterative approximate Byzantine consensus* (IABC) algorithms of interest, and state the validity and convergence conditions that they must satisfy.

Each node i maintains state v_i, with $v_i[t]$ denoting the state of node i at the *end* of the t-th iteration of the algorithm. Initial state of node i, $v_i[0]$, is equal to the initial *input* provided to node i. At the *start* of the t-th iteration ($t > 0$), the state of node i is $v_i[t-1]$. The IABC algorithms of interest will require each node i to perform the following three steps in iteration t, where $t > 0$. Note that the faulty nodes may deviate from this specification.

1. *Transmit step:* Transmit current state, namely $v_i[t-1]$, on all outgoing edges (to nodes in N_i^+).

2. *Receive step:* Receive values on all incoming edges (from nodes in N_i^-). Denote by $r_i[t]$ the vector of values received by node i from its incoming neighbors. If node i does not receive the value from its incoming neighbors, then the message value is assumed to be some *default value.* Therefore, the size of vector $r_i[t]$ is $|N_i^-|$, even if some of node i's incoming neighbors are faulty and choose not to transmit a message.

3. *Update step:* Node i updates its state using a transition function Z_i as follows. Z_i is a part of the specification of the algorithm, and takes as input the vector $r_i[t]$ and state $v_i[t-1]$.

$$v_i[t] \quad = \quad Z_i \ (\ r_i[t] \, , \, v_i[t-1] \ ) \tag{4.1}$$

We now define $U[t]$ and $\mu[t]$, assuming that $\mathcal{F}$ is the set of Byzantine faulty nodes, with the nodes in $\mathcal{V} - \mathcal{F}$ being fault-free.[1]

- $U[t] = \max_{i \in \mathcal{V}-\mathcal{F}} v_i[t]$. $U[t]$ is the largest state among the fault-free nodes at the end of the t-th iteration. Since the initial state of each node is equal to its input, $U[0]$ is equal to the maximum value of the initial input at the fault-free nodes.

- $\mu[t] = \min_{i \in \mathcal{V}-\mathcal{F}} v_i[t]$. $\mu[t]$ is the smallest state among the fault-free nodes at the end of the t-th iteration. $\mu[0]$ is equal to the minimum value of the initial input at the fault-free nodes.

The following conditions must be satisfied by an IABC algorithm in presence of up to f Byzantine faulty nodes:

- *Validity:* $\forall t > 0, \quad \mu[t] \geq \mu[t-1] \quad \text{and} \quad U[t] \leq U[t-1]$

- *Convergence:* $\lim_{t \to \infty} \ U[t] - \mu[t] = 0$

[1] For sets X and Y, $X - Y$ contains elements that are in X but not in Y. That is, $X - Y = \{i \mid i \in X, \ i \notin Y\}$.

The objective in this Chapter is to identify the necessary and sufficient conditions for the existence of a *correct* IABC algorithm (i.e., an algorithm satisfying the above validity and convergence conditions) for a given $G(\mathcal{V}, \mathcal{E})$.

4.3 Necessary Condition

For a correct IABC algorithm to exist, the network graph $G(\mathcal{V}, \mathcal{E})$ must satisfy the necessary condition proved in Chapter 4.3. Theorems 6 and 7 below state equivalent necessary conditions. The form of the necessary condition in Theorem 7 is more intuitive, whereas the form in Theorem 6 is used later to prove sufficiency. We now define relations $\Rightarrow$ and $\not\Rightarrow$ that are used frequently in the discussion in this Chapter. Recall that as defined in Chapter 1.2, we use N_i^- and N_i^+ to represent incoming and outgoing neighbors, respectively.

Definition 10 *For non-empty disjoint sets of nodes A and B,*

- *$A \Rightarrow B$ iff there exists a node $v \in B$ that has at least $f+1$ incoming edges from nodes in A, i.e., $|N_v^- \cap A| > f$.*
- *$A \not\Rightarrow B$ iff $A \Rightarrow B$ is* not *true.*

$\Rightarrow$ is different from $\longmapsto$ defined in Definition 1 of Chapter 3. Roughly speaking, $A \Rightarrow B$ means that each node in B has enough incoming neighbors that are in A, whereas, $A \longmapsto B$ means that all nodes in B *jointly* have enough incoming neighbors that are in A. Also, notice that the condition in the theorem below bears some similarity to Condition BCS in Chapter 3.3. However, the proofs are very different due to the distinct natures of iterative and general algorithms.

Theorem 6 *Suppose that a correct IABC algorithm exists for $G(\mathcal{V}, \mathcal{E})$. Let sets F, L, C, R form a partition of $\mathcal{V}$, such that L and R are both non-empty, and $|F| \le f$. Then, either $C \cup R \Rightarrow L$, or $L \cup C \Rightarrow R$.*

Proof: The proof is by contradiction. Let us assume that a correct iterative consensus algorithm exists, and $C \cup R \not\Rightarrow L$ and $L \cup C \not\Rightarrow R$. Thus, for any $i \in L$, $|N_i^- \cap (C \cup R)| < f+1$, and for any $j \in R$, $|N_j^- \cap (L \cup C)| < f+1$. Figure 4.1 illustrates the sets used in this proof.

Also assume that the nodes in F (if F is non-empty) are all faulty, and the other nodes in sets L, C, R are fault-free. Note that the fault-free nodes are not aware of the identity of the faulty nodes.

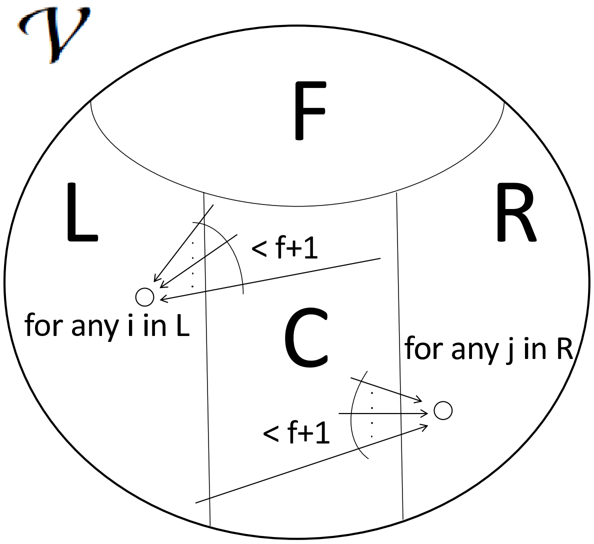

Figure 4.1: Illustration for the proof of Theorem 6. In this figure, $C \cup R \not\Rightarrow L$ and $L \cup C \not\Rightarrow R$.

Consider the case when (i) each node in L has initial input m, (ii) each node in R has initial input M, such that $M > m$, and (iii) each node in C, if C is non-empty, has an input in the interval $[m, M]$.

In the *Transmit Step* of iteration 1, suppose that the faulty nodes in F (if non-empty) send $m^- < m$ on outgoing links to nodes in L, send $M^+ > M$ on outgoing links to nodes in R, and send some arbitrary value in interval $[m, M]$ on outgoing links to the nodes in C (if C is non-empty). This behavior is possible since nodes in F are faulty. Note that $m^- < m < M < M^+$. Each fault-free node $k \in \mathcal{V} - F$, sends to nodes in N_k^+ value $v_k[0]$ in iteration 1.

Consider any node $i \in L$. Denote $N_i' = N_i^- \cap (C \cup R)$. Since $|F| \leq f$, $|N_i^- \cap F| \leq f$. Since $C \cup R \not\Rightarrow L$, $|N_i'| \leq f$. Node i will then receive m^- from the nodes in $N_i^- \cap F$, and values in $[m, M]$ from the nodes in N_i', and m from the nodes in $\{i\} \cup (N_i^- \cap L)$.

Consider the following two cases:

- Both $N_i^- \cap F$ and N_i' are non-empty: Now $|N_i^- \cap F| \leq f$ and $|N_i'| \leq f$. From node i's perspective, consider two possible scenarios: (a) nodes in $N_i^- \cap F$ are faulty, and the other nodes are fault-free, and (b) nodes in N_i' are faulty, and the other nodes are fault-free.

  In scenario (a), from node i's perspective, the faulty-free nodes have sent values in interval $[m, M]$, whereas the faulty nodes have sent value m^-. According to the validity condition, $v_i[1] \geq m$. On the other hand, in scenario (b), the fault-free nodes have sent values m^- and m, where $m^- < m$; so $v_i[1] \leq m$, according to the validity condition. Since node i does not know whether the correct scenario is (a) or (b), it must update its state to satisfy the validity condition in both cases. Thus, it follows that $v_i[1] = m$.

- At most one of $N_i^- \cap F$ and N_i' is non-empty: Thus, $|(N_i^- \cap F) \cup N_i'| \le f$. From node i's perspective, it is possible that the nodes in $(N_i^- \cap F) \cup N_i'$ are all faulty, and the rest of the nodes are fault-free. In this situation, the values sent to node i by the fault-free nodes (which are all in $\{i\} \cup (N_i^- \cap L)$) are all m, and therefore, $v_i[1]$ must be set to m as per the validity condition.

Thus, $v_i[1] = m$ for each node $i \in L$. Similarly, we can show that $v_j[1] = M$ for each node $j \in R$.

Now consider the nodes in set C, if C is non-empty. All the values received by the nodes in C are in $[m, M]$, therefore, their new state must also remain in $[m, M]$, as per the validity condition.

The above discussion implies that, at the end of iteration 1, the following conditions hold true: (i) state of each node in L is m, (ii) state of each node in R is M, and (iii) state of each node in C is in the interval $[m, M]$. These conditions are identical to the initial conditions listed previously. Then, by a repeated application of the above argument (proof by induction), it follows that for any $t \ge 0$, $v_i[t] = m$ for all $\forall i \in L$, $v_j[t] = M$ for all $j \in R$ and $v_k[t] \in [m, M]$ for all $k \in C$.

Since L and R both contain fault-free nodes, the convergence requirement is not satisfied. This is a contradiction to the assumption that a correct iterative algorithm exists. □

Corollary 2 *Suppose that a correct IABC algorithm exists for $G(\mathcal{V}, \mathcal{E})$. Let $\{F, L, R\}$ be a partition of $\mathcal{V}$, such that L and R are both non-empty and $|F| \le f$. Then, either $L \Rightarrow R$ or $R \Rightarrow L$.*

The proof follows by setting $C = \emptyset$ in Theorem 6. Next corollary shows the lower bound on n, the number of nodes in the system.

Corollary 3 *Suppose that a correct IABC algorithm exists for $G(\mathcal{V}, \mathcal{E})$. Then, n must be at least $3f + 1$, and if $f > 0$, then each node must have at least $2f + 1$ incoming edges.*

Proof: The necessary condition of $n \ge 3f + 1$ has been shown previously [32]. We include a proof here for completeness. For $f = 0$, $n \ge 3f + 1$ is trivially true. For $f > 0$, the proof is by contradiction. Suppose that $2 \le n \le 3f$. In this case, we can partition $\mathcal{V}$ into sets L, R, C such that $0 < |L| \le f$, $0 < |R| \le f$ and $0 \le |F| \le f$. Since $0 < |L| \le f$ and $0 < |R| \le f$, we have $L \not\Rightarrow R$ and $R \not\Rightarrow L$, respectively. This violates the necessary condition in Corollary 2.

The proof of the remaining corollary is also by contradiction. Suppose that $f > 0$, and for some node i, $|N_i^-| \le 2f$. Define set $L = \{i\}$. Partition N_i^- into two sets F and H such that $|H| = \lfloor |N_i^-|/2 \rfloor \le f$ and $|F| = \lceil |N_i^-|/2 \rceil \le f$. Define $R = \mathcal{V} - F - L = \mathcal{V} - F - \{i\}$. Since $|\mathcal{V}| = n \ge \max(2, 3f + 1)$, R is non-empty. Now, $N_i^- \cap R = H$, and $|N_i^- \cap R| \le f$. Therefore, since $L = \{i\}$ and $|N_i^- \cap R| \le f$, $R \not\Rightarrow L$. Also, since $|L| = 1 < f + 1$, $L \not\Rightarrow R$. This violates Corollary 2 above. □

Equivalent Condition In Chapter 4.5, we prove that the condition stated in Theorem 6 is also sufficient for the existence of a correct IABC algorithm. The condition in Theorem 6 is not very intuitive and does not provide insights on how to achieve consensus. Thus, in Theorem 7 below, we state another necessary condition that is equivalent to the necessary condition in Theorem 6, and is somewhat easier to interpret. To facilitate the statement of Theorem 7, we now introduce the notion and "reduced graph". Note that it is similar to the one introduced in Chapter 3.6.4. For brevity, we use the same terminology. However, the exact construction of reduced graph below is different from the ones in Chapter 3.6.4. The definition below uses the notions introduced in Definitions 6 and 7 in Chapter 3.6.4.

Definition 11 (Reduced Graph) *For a given graph $G(\mathcal{V}, \mathcal{E})$ and $F \subset \mathcal{V}$, a graph $G_F(\mathcal{V}_F, \mathcal{E}_F)$ is said to be a* reduced graph*, if: (i) $\mathcal{V}_F = \mathcal{V} - F$, and (ii) $\mathcal{E}_F$ is obtained by first removing from $\mathcal{E}$ all the links incident on the nodes in F, and* then *removing up to f other incoming links at each node in $\mathcal{V}_F$.*

Note that for a given $G(\mathcal{V}, \mathcal{E})$ and a given F, multiple reduced graphs G_F may exist.

Theorem 7 *Suppose that the condition in Theorem 6 holds for graph $G(\mathcal{V}, \mathcal{E})$. Then, for any $F \subset \mathcal{V}$ such that $|F| < |\mathcal{V}|$ and $|F| \leq f$, every reduced graph G_F obtained as per Definition 11 must contain exactly one* source component.

Proof: Since $|F| < |\mathcal{V}|$, G_F contains at least one node; therefore, at least one source component must exist in G_F. We now prove that G_F cannot contain more than one source component. The proof is by contradiction. Suppose that there exists a set $F \subset \mathcal{V}$ with $|F| < |\mathcal{V}|$ and $|F| \leq f$, and a reduced graph $G_F(\mathcal{V}_F, \mathcal{E}_F)$ corresponding to F, such that the decomposition of G_F includes at least two source components.

Let the sets of nodes in two such source components of G_F be denoted L and R, respectively. Let $C = \mathcal{V} - F - L - R$. Observe that F, L, C, R form a partition of the nodes in $\mathcal{V}$. Since L is a source component in G_F it follows that there are no directed links in $\mathcal{E}_F$ from any node in $C \cup R$ to the nodes in L. Similarly, since R is a source component in G_F it follows that there are no directed links in $\mathcal{E}_F$ from any node in $L \cup C$ to the nodes in R. These observations, together with the manner in which $\mathcal{E}_F$ is defined, imply that (i) there are at most f links in $\mathcal{E}$ from the nodes in $C \cup R$ to each node in L, and (ii) there are at most f links in $\mathcal{E}$ from the nodes in $L \cup C$ to each node in R. Therefore, in graph $G(\mathcal{V}, \mathcal{E})$, $C \cup R \not\Rightarrow L$ and $L \cup C \not\Rightarrow R$, violating Theorem 6. Thus, we have proved that G_F must contain exactly one source component. □

The above proof shows that the condition in Theorem 6 implies the condition in Theorem 7. Now, we prove the other direction. We achieve this by proving that, if the condition in Theorem 6 does not hold true for $G(\mathcal{V}, \mathcal{E})$, then the condition in Theorem 7 also does not hold true.

Theorem 8 *Suppose that the condition in Theorem 6 does not hold for graph $G(\mathcal{V}, \mathcal{E})$. Then, there exists a reduced graph G_F obtained as per Definition 11 that contains more than one* source component.

Proof: Suppose that the condition stated in Theorem 6 does not hold for $G(\mathcal{V}, \mathcal{E})$. Thus, there exists a partition F, L, C, R of $\mathcal{V}$ such that $|F| \leq f$, L and R are non-empty, and $C \cup R \not\Rightarrow L$ and $L \cup C \not\Rightarrow R$.

We now construct a reduced graph $G_F(\mathcal{V}_F, \mathcal{E}_F)$ corresponding to set F. First, remove all nodes in F from $\mathcal{V}$ to obtain $\mathcal{V}_F$. Remove all the edges incident on F from $\mathcal{E}$. Then because $C \cup R \not\Rightarrow L$, the number of incoming edges at each node in L from the nodes in $C \cup R$ is at most f; remove all these edges. Similarly, for every node $j \in R$, remove all incoming edges from $L \cup C$ (there are at most f such edges at each node $j \in R$). The resulting graph G_F is a reduced graph that satisfies the conditions in Definition 11.

In $\mathcal{E}_F$, there are no incoming edges to nodes in R from the nodes $L \cup C$; similarly, in $\mathcal{E}_F$, there are no incoming edges to nodes L from the nodes in $C \cup R$. It follows that no single node in $\mathcal{V}_F$ has paths in G_F (i.e., paths consisting of links in $\mathcal{E}_F$) to all the other nodes in $\mathcal{V}_F$. Thus, G_F must contain more than one source component. Thus, the condition in Theorem 7 does not hold for $G(\mathcal{V}, \mathcal{E})$. □

Therefore, it follows that Theorems 6 and 7 specify equivalent conditions.[2]

Corollary 4 *Suppose that Theorem 6 holds true for graph $G(\mathcal{V}, \mathcal{E})$. Then, for any $F \subset \mathcal{V}$ such that $|F| \leq f$, the unique source component in every reduced graph G_F must contain at least $f + 1$ nodes.*

Proof: Since the source component is non-empty, the claim is trivially true for $f = 0$.

Now consider $f > 0$. The proof in this case is by contradiction. Suppose that there exists a set F with $|F| \leq f$, and a corresponding reduced graph $G_F(\mathcal{V}_F, \mathcal{E}_F)$, such that the decomposition of G_F contains a unique source component consisting of at most f nodes. Define L to be the set of nodes in this unique source component, and $R = \mathcal{V} - L - F$. Observe that F, L, R form a partition of $\mathcal{V}$. R must contain at least $f + 1$ nodes, since $|L| \leq f$, $|F| \leq f$, and by Corollary 3, $n \geq 3f + 1$.

Since $|L| \leq f$, it follows that in graph $G(\mathcal{V}, \mathcal{E})$, $L \not\Rightarrow R$, Then Corollary 2 implies that, in graph $G(\mathcal{V}, \mathcal{E})$, $R \Rightarrow L$. Thus, there must be a node in L, say node i, that has at least $f + 1$ incoming links in $\mathcal{E}$ from the nodes in R. Since $i \in L$, it follows that $i \notin F$ (by definition of a reduced graph). Also, since i has at least $f + 1$ incoming links in $\mathcal{E}$ from nodes in R, it follows that in $\mathcal{E}_F$, node i must have at least one incoming link from the nodes in R. This contradicts that assumption that set L containing node i is a source component of G_F. □

[2] An alternate interpretation of Theorem 7 is that in graph G_F, non-fault-tolerant iterative consensus must be possible.

4.4 Algorithm 1

We will prove that there exists an IABC algorithm – particularly *Algorithm 1* below – that satisfies the *validity* and *convergence* conditions provided that the graph $G(\mathcal{V}, \mathcal{E})$ satisfies the necessary condition in Theorem 6. This implies that the necessary condition in Theorem 6 is also sufficient. *Algorithm 1* has the three-step structure described in Chapter 4.2, and it is similar to algorithms that were analyzed in prior work as well [29, 52, 41, 49] (although correctness of the algorithm under the necessary condition in Theorem 6 has not been proved previously).

Algorithm 1

1. *Transmit step:* Transmit current state $v_i[t-1]$ on all outgoing edges.

2. Receive step: Receive values on all incoming edges. These values form vector $r_i[t]$ of size $|N_i^-|$. When a fault-free node expects to receive a message from a neighbor but does not receive the message, the message value is assumed to be equal to some *default value*.

3. *Update step:* Sort the values in $r_i[t]$ in an increasing order, and eliminate the smallest f values, and the largest f values (breaking ties arbitrarily). Let $N_i^*[t]$ denote the set of nodes from whom the remaining $|N_i^-| - 2f$ values were received, and let w_j denote the value received from node $j \in N_i^*$. For convenience, define $w_i = v_i[t-1]$ to be the value node i "receives" from itself. Observe that if $j \in \{i\} \cup N_i^*[t]$ is fault-free, then $w_j = v_j[t-1]$.

   Define

   $$v_i[t] \;=\; Z_i(r_i[t], v_i[t-1]) \;=\; \sum_{j \in \{i\} \cup N_i^*[t]} a_i \, w_j \tag{4.2}$$

   where

   $$a_i = \frac{1}{|N_i^-| + 1 - 2f}$$

   Note that $|N_i^*[t]| = |N_i^-| - 2f$, and $i \notin N_i^*[t]$ because $(i,i) \notin \mathcal{E}$. The "weight" of each term on the right-hand side of (4.2) is a_i, and these weights add to 1. Also, $0 < a_i \leq 1$. For future reference, let us define α as:

   $$\alpha = \min_{i \in \mathcal{V}} \; a_i \tag{4.3}$$

4.5 Sufficiency (Correctness of Algorithm 1)

In Theorems 9 and 10 below, we prove that Algorithm 1 satisfies *validity* and *convergence* conditions, respectively, provided that $G(\mathcal{V}, \mathcal{E})$ satisfies the condition below, which matches the necessary condition stated in Theorem 6.

Sufficient condition: For every partition F, L, C, R of $\mathcal{V}$, such that L and R are both non-empty, and $|F| \leq f$, either $C \cup R \Rightarrow L$, or $L \cup C \Rightarrow R$.

Theorem 9 *Suppose that $\mathcal{F}$ is the set of Byzantine faulty nodes, and that $G(\mathcal{V}, \mathcal{E})$ satisfies the* sufficient *condition stated above. Then Algorithm 1 satisfies the* validity *condition.*

Proof: Consider the t-th iteration, and any fault-free node $i \in \mathcal{V} - \mathcal{F}$. Consider two cases:

- $f = 0$: In this case, all nodes must be fault-free, and $\mathcal{F} = \emptyset$. In (4.2) in Algorithm 1, note that $v_i[t]$ is computed using states from the previous iteration at node i and other nodes. By definition of $\mu[t-1]$ and $U[t-1]$, $v_j[t-1] \in [\mu[t-1], U[t-1]]$ for all fault-free nodes $j \in \mathcal{V} - \mathcal{F} = \mathcal{V}$. Thus, in this case, all the values used in computing $v_i[t]$ are in the interval $[\mu[t-1], U[t-1]]$. Since $v_i[t]$ is computed as a weighted average of these values, $v_i[t]$ is also within $[\mu[t-1], U[t-1]]$.
- $f > 0$: By Corollary 3, $|N_i^-| \geq 2f + 1$, and therefore, $|r_i[t]| \geq 2f + 1$. When computing set $N_i^*[t]$, the largest f and smallest f values from $r_i[t]$ are eliminated. Since at most f nodes are faulty, it follows that, either (i) the values received from the faulty nodes are all eliminated, or (ii) the values from the faulty nodes that still remain are between values received from two fault-free nodes. Thus, the remaining values in $r_i[t]$ are all in the interval $[\mu[t-1], U[t-1]]$. Also, $v_i[t-1]$ is in $[\mu[t-1], U[t-1]]$, as per the definition of $\mu[t-1]$ and $U[t-1]$. Thus $v_i[t]$ is computed as a weighted average of values in $[\mu[t-1], U[t-1]]$, and, therefore, it will also be in $[\mu[t-1], U[t-1]]$.

Since $\forall i \in \mathcal{V} - \mathcal{F}$, $v_i[t] \in [\mu[t-1], U[t-1]]$, the validity condition is satisfied. □

Definition 12 *For disjoint sets A, B, $in(A \Rightarrow B)$ denotes the set of all the nodes in B that each have at least $f + 1$ incoming edges from nodes in A. More formally,*

$$in(A \Rightarrow B) = \{\ v \mid v \in B \text{ and } \ f + 1 \leq |N_v^- \cap A|\ \}.$$

With an abuse of notation, when $A \not\Rightarrow B$, define $in(A \Rightarrow B) = \emptyset$.

We introduce a notion of "propagating sequence" below. Note that this is different from the "propagate" notation $\rightsquigarrow$ introduced in Definition 2 in Chapter 3.

Definition 13 ***(Propagating Sequence)*** *For* non-empty disjoint *sets A and B, set A is said to* propagate to *set B in l steps, where $l > 0$, if there exist sequences of sets $A_0, A_1, A_2, \cdots, A_l$ and $B_0, B_1, B_2, \cdots, B_l$ (propagating sequences) such that*

- $A_0 = A, \qquad B_0 = B, \qquad A_l = A \cup B, \qquad B_l = \emptyset, \qquad B_\tau \neq \emptyset$ *for* $\tau < l$, *and*
- *for* $0 \leq \tau \leq l - 1$,
  - $A_\tau \Rightarrow B_\tau$,
  - $A_{\tau+1} = A_\tau \cup in(A_\tau \Rightarrow B_\tau)$, *and*
  - $B_{\tau+1} = B_\tau - in(A_\tau \Rightarrow B_\tau)$

Observe that A_τ and B_τ form a partition of $A \cup B$, and for $\tau < l$, $in(A_\tau \Rightarrow B_\tau) \neq \emptyset$. Also, when set A propagates to set B, the number of steps l in the above definition is upper bounded by $n - f - 1$, since set A must be of size at least $f + 1$ for it to propagate to B; otherwise, $A \not\Rightarrow B$.

Lemma 27 *Assume that $G(\mathcal{V}, \mathcal{E})$ satisfies the* sufficient *condition stated above. For any partition A, B, F of $\mathcal{V}$, where A, B are both non-empty, and $|F| \leq f$, either A propagates to B, or B propagates to A.*

To prove Lemma 27, we first prove the following Lemma.

Lemma 28 *Assume that $G(\mathcal{V}, \mathcal{E})$ satisfies Theorem 6. Consider a partition A, B, F of $\mathcal{V}$ such that A and B are non-empty, and $|F| \leq f$. If $B \not\Rightarrow A$, then set A propagates to set B.*

Proof: Since A, B are non-empty, and $B \not\Rightarrow A$, by Corollary 2, we have $A \Rightarrow B$.

Define $A_0 = A$ and $B_0 = B$. Now, for a suitable $l > 0$, we will build propagating sequences $A_0, A_1, \cdots A_l$ and $B_0, B_1, \cdots B_l$ inductively.

- Recall that $A = A_0$ and $B = B_0 \neq \emptyset$. Since $A \Rightarrow B$, $in(A_0 \Rightarrow B_0) \neq \emptyset$. Define $A_1 = A_0 \cup in(A_0 \Rightarrow B_0)$ and $B_1 = B_0 - in(A_0 \Rightarrow B_0)$.

  If $B_1 = \emptyset$, then $l = 1$, and we have found the propagating sequence already.

  If $B_1 \neq \emptyset$, then define $L = A = A_0$, $R = B_1$ and $C = A_1 - A = B_1 - B$. Since $B \not\Rightarrow A$, $R \cup C \not\Rightarrow L$. Therefore, by Lemma 2, $L \cup C \Rightarrow R$. That is, $A_1 \Rightarrow B_1$.

- For increasing values of $i \geq 0$, given A_i and B_i, where $B_i \neq \emptyset$, by following steps similar to the previous item, we can obtain $A_{i+1} = A_0 \cup in(A_i \Rightarrow B_i)$ and $B_{i+1} = B_i - in(A_i \Rightarrow B_i)$, such that either $B_{i+1} = \emptyset$ or $A_{i+1} \Rightarrow B_{i+1}$.

In the above construction, l is the smallest index such that $B_l = \emptyset$. □

Proof of Lemma 27 Now, we are ready to prove Lemma 27.

Proof: Consider two cases:

- $A \not\Rightarrow B$: Then by Lemma 28 above, B propagates to A, completing the proof.
- $A \Rightarrow B$: In this case, consider two sub-cases:
  - *A propagates to B*: The proof in this case is complete.
  - *A does not propagate to B*: Recall that $A \Rightarrow B$. Since A does not propagate to B, propagating sequences defined in Definition 13 do not exist in this case. More precisely, there must exist $k > 0$, and sets $A_0, A_1, \cdots, A_k$ and $B_0, B_1, \cdots, B_k$, such that:
    - $A_0 = A$ and $B_0 = B$, and
    - for $0 \leq i \leq k-1$,
      - o $A_i \Rightarrow B_i$,
      - o $A_{i+1} = A_i \cup in(A_i \Rightarrow B_i)$, and
      - o $B_{i+1} = B_i - in(A_i \Rightarrow B_i)$.
    - $B_k \neq \emptyset$ <u>and</u> $A_k \not\Rightarrow B_k$.

    The last condition above violates the requirements for A to propagate to B.

    Now, $A_k \neq \emptyset$, $B_k \neq \emptyset$, and A_k, B_k, F form a partition of $\mathcal{V}$. Since $A_k \not\Rightarrow B_k$, by Lemma 28 above, B_k propagates to A_k.

    Given that $B_k \subseteq B_0 = B$, $A = A_0 \subseteq A_k$, and B_k propagates to A_k, now we prove that B propagates to A.

    Recall that A_i and B_i form a partition of $\mathcal{V} - F$.

    Let us define $P = P_0 = B_k$ and $Q = Q_0 = A_k$. Thus, P propagates to Q. Suppose that $P_0, P_1, ...P_m$ and $Q_0, Q_1, \cdots, Q_m$ are the propagating sequences in this case, with P_i and Q_i forming a partition of $P \cup Q = A_k \cup B_k = \mathcal{V} - F$.

Let us define $R = R_0 = B$ and $S = S_0 = A$. Note that R, S form a partition of $A \cup B = \mathcal{V} - F$. Now, $P_0 = B_k \subseteq B = R_0$ and $S_0 = A \subseteq A_k = Q_0$. Also, $R_0 - P_0$ and S_0 form a partition of Q_0. Figure 4.2 illustrates some of the sets used in this proof.

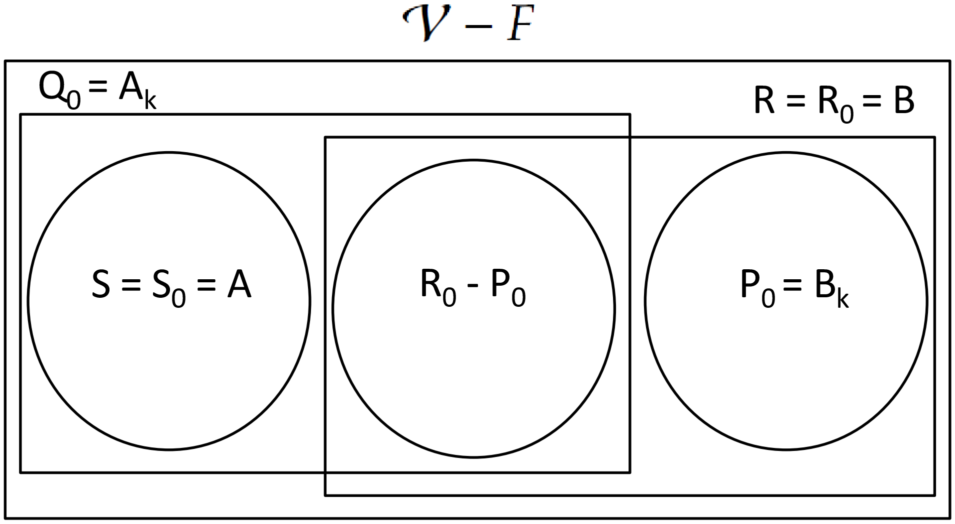

Figure 4.2: Illustration for the last part of the proof of Lemma 27. In this figure, $R_0 = P_0 \cup (R_0 - P_0)$ and $Q_0 = S_0 \cup (R_0 - P_0)$.

- $*$ Define $P_1 = P_0 \cup (in(P_0 \Rightarrow Q_0))$, and $Q_1 = \mathcal{V} - F - P_1 = Q_0 - (in(P_0 \Rightarrow Q_0))$. Also, $R_1 = R_0 \cup (in(R_0 \Rightarrow S_0))$, and $S_1 = \mathcal{V} - F - R_1 = S_0 - (in(R_0 \Rightarrow S_0))$.

  Since $R_0 - P_0$ and S_0 are a partition of Q_0, the nodes in $in(P_0 \Rightarrow Q_0)$ belong to one of these two sets. Note that $R_0 - P_0 \subseteq R_0$. Also, $S_0 \cap in(P_0 \Rightarrow Q_0) \subseteq in(R_0 \Rightarrow S_0)$. Therefore, it follows that $P_1 = P_0 \cup (in(P_0 \Rightarrow Q_0)) \subseteq R_0 \cup (in(R_0 \Rightarrow S_0)) = R_1$.

  Thus, we have shown that, $P_1 \subseteq R_1$. Then it follows that $S_1 \subseteq Q_1$.

- $*$ For $0 \leq i < m$, let us define $R_{i+1} = R_i \cup in(R_i \Rightarrow S_i)$ and $S_{i+1} = S_i - in(R_i \Rightarrow S_i)$. Then following an argument similar to the above case, we can inductively show that, $P_i \subseteq R_i$ and $S_i \subseteq Q_i$. Due to the assumption on the length of the propagating sequence above, $P_m = P \cup Q = \mathcal{V} - F$ and $Q_m = \emptyset$. Thus, there must exist $r \leq m$, such that for $i < r$,

$R_i \neq \mathcal{V} - F$, and $R_r = \mathcal{V} - F$ and $S_r = \emptyset$.

The sequences $R_0, R_1, \cdots, R_r$ and $S_0, S_1, \cdots, S_r$ form propagating sequences, proving that $R = B$ propagates to $S = A$.

□

The lemma below states that the interval to which the states at all the fault-free nodes are confined shrinks after a finite number of iterations of Algorithm 1. Recall that $U[t]$ and $\mu[t]$ (defined in Chapter 4.2) are the maximum and minimum over the states at the fault-free nodes at the end of the t-th iteration.

Lemma 29 *Suppose that $G(\mathcal{V}, \mathcal{E})$ satisfies the* sufficient *condition stated above, and $\mathcal{F}$ is the set of Byzantine faulty nodes. Moreover, at the end of the s-th iteration of Algorithm 1, suppose that the fault-free nodes in $\mathcal{V} - \mathcal{F}$ can be partitioned into non-empty sets R and L such that (i) R propagates to L in l steps, and (ii) the states of nodes in R are confined to an interval of length $\leq \frac{U[s]-\mu[s]}{2}$. Then, with Algorithm 1,*

$$U[s+l] - \mu[s+l] \;\leq\; \left(1 - \frac{\alpha^l}{2}\right)(U[s] - \mu[s]) \tag{4.4}$$

where α is as defined in (4.3).

We first present two additional lemmas (using the notation in Algorithm 1).

Lemma 30 *Suppose that $\mathcal{F}$ is the set of faulty nodes, and that $G(\mathcal{V}, \mathcal{E})$ satisfies the "sufficient condition" stated in Chapter 4.5. Consider node $i \in \mathcal{V} - \mathcal{F}$. Let $\psi \leq \mu[t-1]$. Then, for $j \in \{i\} \cup N_i^*[t]$,*

$$v_i[t] - \psi \;\geq\; a_i\ (w_j - \psi)$$

Specifically, for fault-free $j \in \{i\} \cup N_i^*[t]$,

$$v_i[t] - \psi \;\geq\; a_i\ (v_j[t-1] - \psi)$$

Proof: In (4.2) in Algorithm 1, for each $j \in \{i\} \cup N_i^*[t]$, consider two cases:

- j is faulty-free: Then, either $j = i$ or $j \in N_i^*[t] \cap (\mathcal{V} - \mathcal{F})$. In this case, $w_j = v_j[t-1]$. Therefore, $\mu[t-1] \leq w_j \leq U[t-1]$.
- j is faulty: In this case, f must be non-zero (otherwise, all nodes are fault-free). By Corollary 3, $|N_i^-| \geq 2f + 1$. Then it follows that, in step 2 of Algorithm 1, the smallest f values in $r_i[t]$ contain

the state of at least one fault-free node, say k. This implies that $v_k[t-1] \leq w_j$. This, in turn, implies that $\mu[t-1] \leq w_j$.

Thus, for all $j \in \{i\} \cup N_i^*[t]$, we have $\mu[t-1] \leq w_j$. Therefore,

$$w_j - \psi \geq 0 \text{ for all } j \in \{i\} \cup N_i^*[t] \tag{4.5}$$

Since weights in (4.2) in Algorithm 1 add to 1, we can re-write that equation as,

$$\begin{aligned} v_i[t] - \psi &= \sum_{j \in \{i\} \cup N_i^*[t]} a_i\,(w_j - \psi) \qquad (4.6) \\ &\geq a_i\,(w_j - \psi), \quad \forall j \in \{i\} \cup N_i^*[t] \qquad \text{from (4.5)} \end{aligned}$$

For fault-free $j \in \{i\} \cup N_i^*[t]$, $w_j = v_j[t-1]$, therefore,

$$v_i[t] - \psi \quad \geq \quad a_i\,(v_j[t-1] - \psi) \tag{4.7}$$

□

Lemma 31 *Suppose that $\mathcal{F}$ is the set of faulty nodes, and that $G(\mathcal{V}, \mathcal{E})$ satisfies the "sufficient condition" stated in Chapter 4.5. Consider fault-free node $i \in \mathcal{V} - \mathcal{F}$. Let $\Psi \geq U[t-1]$. Then, for $j \in \{i\} \cup N_i^*[t]$,*

$$\Psi - v_i[t] \geq a_i\ (\Psi - w_j)$$

Specifically, for fault-free $j \in \{i\} \cup N_i^*[t]$,

$$\Psi - v_i[t] \geq a_i\ (\Psi - v_j[t-1])$$

Proof: The proof is similar to Lemma 30 proof. □

Proof of Lemma 29

Next, we will use Lemmas 30 and 31 to show that Lemma 29 holds.

Proof: Since R propagates to L, as per Definition 13, there exist sequences of sets $R_0, R_1, \cdots, R_l$ and $L_0, L_1, \cdots, L_l$, where

- $R_0 = R, \quad L_0 = L, \quad R_l = R \cup L, \quad L_l = \emptyset, \quad$ for $0 \leq \tau < l$, $L_\tau \neq \emptyset$, and

- for $0 \le \tau \le l-1$,
    - $R_\tau \Rightarrow L_\tau$,
    - $R_{\tau+1} = R_\tau \cup in(R_\tau \Rightarrow L_\tau)$, and
    - $L_{\tau+1} = L_\tau - in(R_\tau \Rightarrow L_\tau)$

Let us define the following bounds on the states of the nodes in R at the end of the s-th iteration:

$$\begin{aligned} M &= max_{j \in R}\ v_j[s] && (4.8) \\ m &= min_{j \in R}\ v_j[s] && (4.9) \end{aligned}$$

By the assumption in the statement of Lemma 29,

$$M - m \le \frac{U[s] - \mu[s]}{2} \tag{4.10}$$

Also, $M \le U[s]$ and $m \ge \mu[s]$. Therefore, $U[s] - M \ge 0$ and $m - \mu[s] \ge 0$.

The remaining proof of Lemma 29 relies on derivation of the three intermediate claims below.

Claim 4 *For $0 \le \tau \le l$, for each node $i \in R_\tau$,*

$$v_i[s+\tau] - \mu[s] \ \ge\ \alpha^\tau (m - \mu[s]) \tag{4.11}$$

Proof of Claim 4: The proof is by induction.

Induction basis: By definition of m, (4.11) holds true for $\tau = 0$.

Induction: Assume that (4.11) holds true for some τ, $0 \le \tau < l$. Consider $R_{\tau+1}$. Observe that R_τ and $R_{\tau+1} - R_\tau$ form a partition of $R_{\tau+1}$; let us consider each of these sets separately.

- Set R_τ: By assumption, for each $i \in R_\tau$, (4.11) holds true. By validity of Algorithm 1 (proved in Theorem 9), $\mu[s] \le \mu[s+\tau]$. Therefore, setting $\psi = \mu[s]$ and $t = s + \tau + 1$ in Lemma 30, we get,

$$\begin{aligned} v_i[s+\tau+1] - \mu[s] &\ge a_i\ (v_i[s+\tau] - \mu[s]) \\ &\ge a_i\ \alpha^\tau (m - \mu[s]) \quad \text{due to (4.11)} \\ &\ge \alpha^{\tau+1}(m - \mu[s]) \quad \text{due to (4.3)} \\ &\qquad \text{and because} \quad m - \mu[s] \ge 0 \end{aligned}$$

- Set $R_{\tau+1} - R_\tau$: Consider a node $i \in R_{\tau+1} - R_\tau$. By definition of $R_{\tau+1}$, we have that $i \in in(R_\tau \Rightarrow L_\tau)$. Thus,

  $$|N_i^- \cap R_\tau| \geq f + 1$$

  In Algorithm 1, $2f$ values (f smallest and f largest) received by node i are eliminated before $v_i[s+\tau+1]$ is computed at the end of $(s + \tau + 1)$-th iteration. Consider two possibilities:

  – Value received from one of the nodes in $N_i^- \cap R_\tau$ is *not* eliminated. Suppose that this value is received from fault-free node $p \in N_i^- \cap R_\tau$. Then, by an argument similar to the previous case, we can set $\psi = \mu[s]$ in Lemma 30, to obtain,

  $$\begin{aligned} v_i[s + \tau + 1] - \mu[s] &\geq a_i\ (v_p[s + \tau] - \mu[s]) \\ &\geq a_i\ \alpha^\tau(m - \mu[s]) \quad \text{due to (4.11)} \\ &\geq \alpha^{\tau+1}(m - \mu[s]) \quad \text{due to (4.3)} \\ &\quad\ \text{and because} \quad m - \mu[s] \geq 0 \end{aligned}$$

  – Values received from *all* (there are at least $f + 1$) nodes in $N_i^- \cap R_\tau$ are eliminated. Note that in this case f must be non-zero (for $f = 0$, no value is eliminated, as already considered in the previous case). By Corollary 3, we know that each node must have at least $2f+1$ incoming edges. Since at least $f + 1$ values from nodes in $N_i^- \cap R_\tau$ are eliminated, and there are at least $2f + 1$ values to choose from, it follows that the values that are *not* eliminated[3] are within the interval to which the values from $N_i^- \cap R_\tau$ belong. Thus, there exists a node k (possibly faulty) from whom node i receives some value w_k – which is not eliminated – and a fault-free node $p \in N_i^- \cap R_\tau$ such that

  $$v_p[s + \tau] \quad \leq \quad w_k \tag{4.12}$$

[3] At least one value received from the nodes in N_i^- is not eliminated, since there are $2f + 1$ incoming edges, and only $2f$ values are eliminated.

Then by setting $\psi = \mu[s]$ and $t = s + \tau + 1$ in Lemma 30, we have

$$\begin{aligned}
v_i[s+\tau+1] - \mu[s] &\geq a_i\ (w_k - \mu[s]) \\
&\geq a_i\ (v_p[s+\tau] - \mu[s]) \quad \text{by (4.12)} \\
&\geq a_i\ \alpha^{\tau}(m - \mu[s]) \quad \text{due to (4.11)} \\
&\geq \alpha^{\tau+1}(m - \mu[s]) \quad \text{due to (4.3)} \\
&\quad\ \text{and because} \quad m - \mu[s] \geq 0
\end{aligned}$$

Thus, we have shown that for all nodes in $R_{\tau+1}$,

$$v_i[s+\tau+1] - \mu[s] \geq \alpha^{\tau+1}(m - \mu[s])$$

This completes the proof of Claim 4.

Claim 5 *For each node $i \in \mathcal{V} - \mathcal{F}$,*

$$v_i[s+l] - \mu[s] \ \geq \ \alpha^l(m - \mu[s]) \tag{4.13}$$

Proof of Claim 5: Note that by definition, $R_l = \mathcal{V} - \mathcal{F}$. Then the proof follows by setting $\tau = l$ in the above Claim 4.

Claim 6 *For each node $i \in \mathcal{V} - \mathcal{F}$,*

$$U[s] - v_i[s+l] \geq \alpha^l(U[s] - M) \tag{4.14}$$

The proof of Claim 6 is similar to the proof of Claim 5.

Now let us resume the proof of the Lemma 29. Note that $R_l = \mathcal{V} - \mathcal{F}$. Thus,

$$\begin{aligned}
U[s+l] &= \max_{i \in \mathcal{V}-\mathcal{F}} v_i[s+l] \\
&\leq U[s] - \alpha^l(U[s] - M) \qquad \text{by (4.14)} \qquad (4.15)
\end{aligned}$$

and

$$\begin{aligned} \mu[s+l] &= \min_{i \in \mathcal{V}-\mathcal{F}} v_i[s+l] \\ &\geq \mu[s] + \alpha^l(m - \mu[s]) \qquad \text{by (4.13)} \end{aligned} \tag{4.16}$$

Subtracting (4.16) from (4.15),

$$\begin{aligned} & U[s+l] - \mu[s+l] \\ \leq\ & U[s] - \alpha^l(U[s] - M) - \mu[s] - \alpha^l(m - \mu[s]) \\ =\ & (1-\alpha^l)(U[s] - \mu[s]) + \alpha^l(M - m) \\ \leq\ & (1-\alpha^l)(U[s] - \mu[s]) + \alpha^l\ \frac{U[s] - \mu[s]}{2} \qquad \text{by (4.10)} \\ \leq\ & (1 - \frac{\alpha^l}{2})(U[s] - \mu[s]) \end{aligned}$$

This concludes the proof of Lemma 29. □

Theorem 10 *Suppose that $\mathcal{F}$ is the set of Byzantine faulty nodes, and that $G(\mathcal{V}, \mathcal{E})$ satisfies the* sufficient *condition stated above. Then Algorithm 1 satisfies the* convergence *condition.*

Proof: Our goal is to prove that, given any $\epsilon > 0$, there exists τ such that

$$U[t] - \mu[t] \leq \epsilon \quad \forall t \geq \tau \tag{4.17}$$

Consider s-th iteration, for some $s \geq 0$. If $U[s] - \mu[s] = 0$, then the algorithm has already converged, and the proof is complete, with $\tau = s$ (recall that we have already proved that the algorithm satisfies the validity condition).

Now consider the case when $U[s] - \mu[s] > 0$. Partition $\mathcal{V} - \mathcal{F}$ into two subsets, A and B, such that, for each node $i \in A$, $v_i[s] \in \left[\mu[s], \frac{U[s]+\mu[s]}{2}\right)$, and for each node $j \in B$, $v_j[s] \in \left[\frac{U[s]+\mu[s]}{2}, U[s]\right]$. By definition of $\mu[s]$ and $U[s]$, there exist fault-free nodes i and j such that $v_i[s] = \mu[s]$ and $v_j[s] = U[s]$. Thus, sets A and B are both non-empty. By Lemma 27, one of the following two conditions must be true:

- Set A propagates to set B. Then, define $L = B$ and $R = A$. The states of all the nodes in $R = A$ are confined within an interval of length $< \frac{U[s]+\mu[s]}{2} - \mu[s] \leq \frac{U[s]-\mu[s]}{2}$.

- Set B propagates to set A. Then, define $L = A$ and $R = B$. In this case, states of all the nodes in $R = B$ are confined within an interval of length $\leq U[s] - \frac{U[s]+\mu[s]}{2} \leq \frac{U[s]-\mu[s]}{2}$.

In both cases above, we have found non-empty sets L and R such that (i) L, R is a partition of $\mathcal{V} - \mathcal{F}$, (ii) R propagates to L, and (iii) the states in R are confined to an interval of length $\leq \frac{U[s]-\mu[s]}{2}$. Suppose that R propagates to L in $l(s)$ steps, where $l(s) \geq 1$. Then by Lemma 29,

$$U[s+l(s)] - \mu[s+l(s)] \leq \left(1 - \frac{\alpha^{l(s)}}{2}\right)(U[s] - \mu[s]) \tag{4.18}$$

In Algorithm 1, observe that $a_i > 0$ for all i. Therefore, α defined in (4.3) in Algorithm 1 is > 0. Then, $n - f - 1 \geq l(s) \geq 1$ and $0 < \alpha \leq 1$; hence, $0 \leq \left(1 - \frac{\alpha^{l(s)}}{2}\right) < 1$.

Let us define the following sequence of iteration indices:

- $\tau_0 = 0$,
- for $i > 0$, $\tau_i = \tau_{i-1} + l(\tau_{i-1})$, where $l(s)$ for any given s was defined above.

If for some i, $U[\tau_i] - \mu[\tau_i] = 0$, then since the algorithm is already proved to satisfy the validity condition, we will have $U[t] - \mu[t] = 0$ for all $t \geq \tau_i$, and the proof of convergence is complete.

Now suppose that $U[\tau_i] - \mu[\tau_i] \neq 0$ for the values of i in the analysis below. By repeated application of the argument leading to (4.18), we can prove that, for $i \geq 0$,

$$U[\tau_i] - \mu[\tau_i] \leq \left(\Pi_{j=1}^{i}\left(1 - \frac{\alpha^{\tau_j - \tau_{j-1}}}{2}\right)\right)(U[0] - \mu[0]) \tag{4.19}$$

For a given ϵ, by choosing a large enough i, we can obtain

$$\left(\Pi_{j=1}^{i}\left(1 - \frac{\alpha^{\tau_j - \tau_{j-1}}}{2}\right)\right)(U[0] - \mu[0]) \leq \epsilon$$

and, therefore,

$$U[\tau_i] - \mu[\tau_i] \leq \epsilon \tag{4.20}$$

For $t \geq \tau_i$, by validity of Algorithm 1, it follows that

$$U[t] - \mu[t] \leq U[\tau_i] - \mu[\tau_i] \leq \epsilon$$

This concludes the proof of Theorem 10. □

It should be easy to see that other correct IABC algorithms can be obtained by choosing "weights" differently than the ones in Algorithm 1, and with other appropriate ways of eliminating values in the *Update step*. This observation inspire us to design IABC algorithms that tolerate other kind of fault models as discussed in later Chapters.

4.6 Asynchronous Systems

Dolev et al. [29] proposed an iterative algorithm for asynchronous systems. We extend their approach to arbitrary point-to-point networks. In particular, we consider the *Asynchronous IABC Algorithm* structure below, which is similar to the algorithm in [29]. This algorithm structure differs from the structure presented in Chapter 4.2 in two important ways: (i) the messages containing states are now tagged by the iteration index to which the states correspond, and (ii) each node i waits to receive only $|N_i^-| - f$ messages containing states from iteration $t-1$ before computing the new state in its t-th iteration. Due to the asynchronous nature of the system, different nodes may potentially perform their t-th iteration at very different real times.

Asynchronous IABC Algorithm

Steps that should be performed by each node $i \in \mathcal{V}$ in its t-th iteration are as follows.

1. *Transmit step:* Transmit current state $v_i[t-1]$ on all outgoing edges. The message is tagged by index $t-1$.

2. *Receive step:* Wait until $|N_i^-| - f$ messages tagged by index $t-1$ are received on the incoming edges. Values received in these messages form vector $r_i[t]$ of size $|N_i^-| - f$. Note that by assumption, up to f nodes may be faulty. Thus, node i can receive $|N_i^-| - f$ messages with index $t-1$.

3. *Update step:* Node i updates its state using a transition function Z_i.

$$v_i[t] \quad = \quad Z_i \ (\ r_i[t] \, , \, v_i[t-1] \ ) \tag{4.21}$$

We now introduce relation $\overset{a}{\Rightarrow}$ that is analogous to relation $\Rightarrow$ defined previously.

Definition 14 *For non-empty disjoint sets of nodes A and B, $A \overset{a}{\Rightarrow} B$ iff there exists a node $v \in B$ that has at least $2f+1$ incoming edges from nodes in A, i.e., $|N_v^- \cap A| \geq 2f+1$.*

Theorem 11 states a necessary condition for asynchronous iterative algorithms with the above structure.

Theorem 11 *If an* Asynchronous IABC Algorithm *satisfies* validity *and* convergence *conditions in graph* $G(\mathcal{V}, \mathcal{E})$*, then for any partition* F, L, C, R *of* $\mathcal{V}$*, such that* L *and* R *are both non-empty and* $|F| \leq f$*, then either* $C \cup R \overset{a}{\Rightarrow} L$*, or* $L \cup C \overset{a}{\Rightarrow} R$*.*

The proof is similar to the proof of Theorem 6. We present the details in Appendix A. The following corollary can be obtained from Theorem 11 as shown in Appendix A.

Corollary 5 *If an* Asynchronous IABC Algorithm *satisfies* validity *and* convergence *conditions in graph* $G(\mathcal{V}, \mathcal{E})$*, then* $n > 5f$*, and when* $f > 0$*,* $|N_i^-| \geq 3f + 1$ *for all* $i \in \mathcal{V}$*.*

It can be shown that the necessary condition in Theorem 11 is also sufficient. In particular, an *Asynchronous IABC Algorithm* with the structure above that performs the *Update step* shown below can be proved to satisfy the convergence and validity conditions. Note that the *Update step* below, to be performed by each node $i \in \mathcal{V}$, is similar to that in *Algorithm 1* for the synchronous system.

- *Update step:* Sort the values in vector $r_i[t]$ in an increasing order, and eliminate the smallest f and the largest f values (breaking ties arbitrarily). Recall that $r_i[t]$ contains $|N_i^-| - f$ values. Let $N_i^*[t]$ denote the set of nodes from whom the remaining $|N_i^-| - 3f$ values were received, and let w_j denote the value received from node $j \in N_i^*[t]$. Define $w_i = v_i[t-1]$, and

  $$v_i[t] \quad = \sum_{j \in \{i\} \cup N_i^*[t]} a_i \, w_j \tag{4.22}$$

  where

  $$a_i = \frac{1}{|N_i^-| + 1 - 3f}.$$

4.7 Summary

This Chapter presents the joint work with Nitin H. Vaidya and Guanfeng Liang – a *tight* necessary and sufficient condition for the existence of a class of synchronous iterative approximate Byzantine consensus algorithms (IABC) that can tolerate up to f Byzantine fault (f-total model) in arbitrary directed graphs. These results are also extended to a class of iterative algorithms for asynchronous systems.

Chapter 5

Iterative Approximate Byzantine Consensus Under Generalized Faults

5.1 Introduction

Chapter 4 discusses the results related to IABC algorithms under f-total fault model in both synchronous and asynchronous systems. This Chapter generalizes the results to a more general fault model in synchronous systems. The fault model assumed in much of the work on Byzantine consensus allows up to f Byzantine faulty nodes in the network (f-total fault model) [94, 49, 46, 29, 52]. In prior work, other fault models have been explored as well. For instance, in the f-local fault model, up to f incoming neighbors of each fault-free node in the network may be faulty [10, 42, 94, 49], and in the‘f-fraction model [94, 49], up to f fraction of incoming neighbors of each fault-free node may be faulty. In this Chapter, we consider a *generalized* fault model (to be formally described in Chapter 5.2), which is similar to the fault model presented in [39], [43]. The *generalized fault* model specifies a “fault domain”, which is a collection of feasible fault sets. For example, in a system consisting of four nodes, namely, nodes $1, 2, 3$ and 4, the fault domain could be specified as $\mathcal{F} = \{ \{1\}, \{2, 3, 4\} \}$. Thus, in this case, either node 1 may be faulty, or any subset of nodes in $\{2, 3, 4\}$ may be faulty. However, node 1 may not be faulty together with another node in the same execution. This fault model is general in the sense that the other fault models, such as f-total, f-local and f-fraction models, are special cases of the generalized fault model.

In this Chapter, we consider iterative algorithms for achieving approximate Byzantine consensus in synchronous point-to-point networks that are modeled as an arbitrary *directed* graphs. The *iterative approximate Byzantine consensus* (IABC) algorithms of interest is discussed in Chapter 4.2 and have the following properties:

- *Initial state* of each node is equal to a real-valued *input* provided to that node.
- *Validity* condition: After each iteration of an IABC algorithm, the state of each fault-free node must remain in the *convex hull* of the states of the fault-free nodes at the end of the *previous* iteration.
- *Convergence* condition: For any $\epsilon > 0$, after a sufficiently large number of iterations, the states of the

fault-free nodes are guaranteed to be within ϵ of each other.

This Chapter is a generalization of Chapter 4 and has two contributions:

- We identify a *tight* necessary condition (Chapter 5.3) on the communication graph for the existence of a correct IABC algorithm under the *generalized* fault model. Moreover, we show that the necessary condition is also sufficient by introducing a new IABC algorithm for the generalized fault model (Chapter 5.4) that uses only "local" information.

- We present a *transition matrix* representation of the new IABC algorithm (Chapter 5.5). This representation is then used to prove the correctness of the proposed algorithm (Chapter 5.5.4). Transition matrices have been used previously to prove correctness of non-fault-tolerant consensus [38]. However, this Chapter is the first to develop transition matrix representation for Byzantine fault-tolerant consensus. We make the following observation: for a given evolution of the state vector corresponding to the state of the fault-free nodes, many alternate state transition matrices may potentially be chosen to emulate that evolution correctly. However, for any given state evolution, we can suitably "design" the transition matrices so that the classical results on matrix products can be applied to prove convergence of our algorithm in all networks that satisfy the necessary condition.

The results presented in this Chapter are published in [84].

5.2 Generalized Byzantine Fault Model

The system and fault models under consideration are stated in Chapter 1.2. We also consider the iterative approximate Byzantine consensus (IABC) Algorithms in this Chapter. The IABC algorithm is described in Chapter 4.2.

The generalized fault model we consider is similar to fault models presented in [39, 43]. The generalized fault model is characterized using *fault domain* $\mathcal{F} \subseteq 2^{\mathcal{V}}$ as follows: Nodes in set F may fail during an execution of the algorithm only if there exists set $F^* \in \mathcal{F}$ such that $F \subseteq F^*$. Set F is then said to be a *feasible* fault set.

Definition 15 *Set $F \subseteq \mathcal{V}$ is said to be a* <u>*feasible*</u> *fault set, if there exists $F^* \in \mathcal{F}$ such that $F \subseteq F^*$.*

Thus, each set in $\mathcal{F}$ specifies nodes that may all potentially fail during a single execution of the algorithm. This feature can be used to capture the notion of correlated failures. For example, consider a system consisting of four nodes, namely, nodes 1, 2, 3, and 4. Suppose that

$$\mathcal{F} = \{ \{1\}, \{2\}, \{3, 4\} \}$$

This definition of $\mathcal{F}$ implies that during an execution either (i) node 1 may fail, or (ii) node 2 may fail, or (iii) any subset of $\{3, 4\}$ may fail, and no other combination of nodes may fail (e.g., nodes 1 and 3 cannot both fail in a single execution). In this case, the reason that the set $\{3, 4\}$ is in the fault domain may be that the failures of nodes 3 and 4 are correlated.

The generalized fault model is also useful to capture variations in node reliability [39, 43]. For instance, in the above example, nodes 1 and 2 may be more reliable than nodes 3 and 4. Therefore, while nodes 3 and 4 may fail in the same execution, nodes 1 and 2 are less likely to fail together in the same execution. Therefore, $\{1, 2\} \not\in \mathcal{F}$.

Local knowledge of $\mathcal{F}$: To implement our IABC Algorithm presented in Chapter 5.4, it is sufficient for each node i to know $N_i^- \cap F$, for each feasible fault set F. In other words, each node only needs to know the set of its incoming neighbors that may fail in the same execution of the algorithm. Thus, the iterative algorithm can be implemented using only "local" information regarding $\mathcal{F}$.

5.3 Necessary Condition

In this section, we develop a necessary condition for the existence of a correct IABC algorithm under generalized fault model. The necessary condition will be proved to be also sufficient in Chapter 5.5. To facilitate the statement, we introduce the notion of "reduced graph", which is based on the notion of graph decomposition and source component introduced in Chapter 3.6.4. The following definition is a generalization of Definition 11 in Chapter 4. For brevity, we use the same terminology.

Definition 16 (Reduced Graph) *For a given graph $G(\mathcal{V}, \mathcal{E})$ and a feasible fault set F, a reduced graph $G_F(\mathcal{V}_F, \mathcal{E}_F)$ is obtained as follows:*

- *Node set is obtained as $\mathcal{V}_F = \mathcal{V} - F$.*
- *For each node $i \in \mathcal{V}_F$, a feasible fault set $F_x(i)$ is chosen, and then the edge set $\mathcal{E}_F$ is obtained as follows:*
    - *remove from $\mathcal{E}$ all the links incident on the nodes in F, i.e., all the incoming and outgoing links of nodes in F, and*

– *for each $j \in F_x(i) \cap \mathcal{V}_F \cap N_i^-$, remove link (j, i) from $\mathcal{E}$.*

Feasible fault sets $F_x(i)$ and $F_x(j)$ chosen for $i \neq j$ may or may not be identical.

Note that for a given $G(\mathcal{V}, \mathcal{E})$ and a given F, multiple reduced graphs G_F may exist, depending on the choice of F_x sets above. The above definition of a "reduced graph" is a generalization of Definition 11 in Chapter 4. Definition 11 only requires to remove enough number of edges while constructing reduced graph. However, in the generalized fault model, we can only delete those edges between node i and some nodes belonging to feasible fault sets $(F_x(i) \cap \mathcal{V}_F)$ to form a "reduced graph". Intuitively, this represents the gain from the extra information, since node i now knows which information are more reliable.

For a correct IABC algorithm to exist, the network graph $G(\mathcal{V}, \mathcal{E})$ must satisfy the necessary condition stated in Theorem 12 below. The proof is similar to the necessity proof in Chapter 4.3.

Theorem 12 *Suppose that a correct IABC algorithm exists for $G(\mathcal{V}, \mathcal{E})$. Then, any reduced graph G_F, corresponding to any feasible fault set F, must contain exactly one* source component.

Proof: Now, we present the proof for Theorem 12. The proof is by contradiction. Let us assume that a correct IABC algorithm exists, and for some feasible fault set F, and feasible sets $F_x(i)$ for each $i \in \mathcal{V} - F$, the resulting reduced graph contains two source components.

Let L and R denote the nodes in the two source components, respectively. Thus, L and R are disjoint and non-empty. Let $C = (\mathcal{V} - F - L - R)$ be the remaining nodes in the reduced graph. C may or may not be non-empty. Let us now assume that the nodes in F (if non-empty) are all faulty, and all the nodes in L, R, and C (if non-empty) are fault-free.

Consider the case when (i) each node in L has initial input m , (ii) each node in R has initial input M, such that $M > m$, and (iii) each node in C (if non-empty) has an input in the interval $[m, M]$.

In the *Transmit step* of iteration 1 of the IABC algorithm, suppose that the faulty nodes in F (if non-empty) send $m^- < m$ on outgoing links to nodes in L, send $M^+ > M$ on outgoing links to nodes in R, and send some arbitrary value in interval $[m, M]$ on outgoing links to nodes in C (if non-empty). This behavior is possible since nodes in F are Byzantine faulty. Note that $m^- < m < M < M^+$. Each fault-free node $k \in \mathcal{V} - F$ sends to nodes in N_k^+ value $v_k[0]$ in iteration 1.

Consider any node $i \in L$. Since L is a source component in the reduced graph, it must be true that $N_i^- \cap (C \cup R) \subseteq N_i^- \cap F_x(i) \cap \mathcal{V}_F$.[1]

[1] Explanation: In the reduced graph, there are no incoming links at i from nodes in $N_i^- \cap (C \cup R)$. Thus, any incoming links in $\mathcal{E}$ from the nodes in $N_i^- \cap (C \cup R)$ must have been removed when constructing $\mathcal{E}_F$ for the reduced graph. Recall that when constructing $\mathcal{E}_F$, incoming links from nodes in $N_i^- \cap F_x(i) \cap \mathcal{V}_F$ are removed. It should be noted that the algorithm is performed using the links in $\mathcal{E}$, not the reduced graph. Thus, in the *Transmit step*, all links in $\mathcal{E}$ are used.

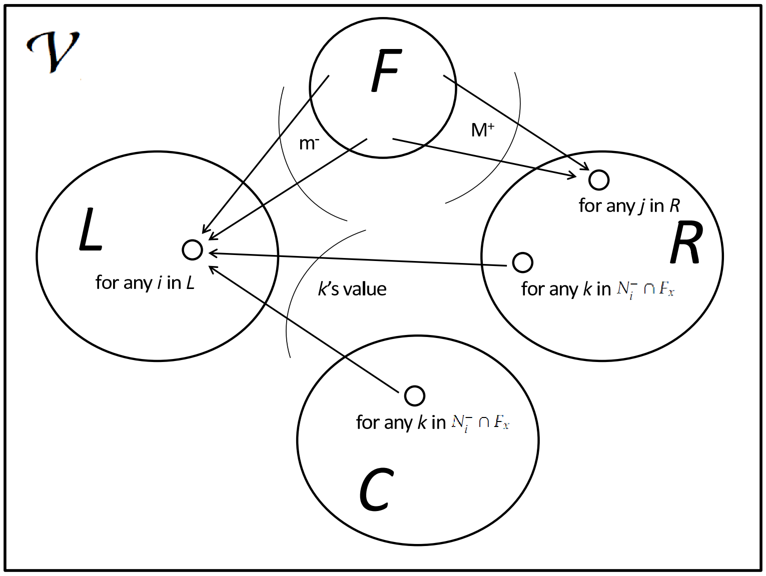

Figure 5.1: Illustration of the behavior of faulty nodes in F and the value received at node i.

Now, node i receives m^- from the nodes in $N_i^- \cap F$, and values in $[m, M]$ from the nodes in $N_i^- \cap (C \cup R)$, and m from the nodes in $\{i\} \cup (N_i^- \cap L)$. Figure 5.1 illustrates the behavior of faulty nodes in F and the value received by node i.

Consider the following two cases:

- **$N_i^- \cap F$ and $N_i^- \cap (C \cup R)$ are both non-empty:** In this case, $(N_i^- \cap F) \subseteq F$ and $N_i^- \cap (C \cup R) = N_i^- \cap F_x(i) \cap \mathcal{V}_F \subseteq F_x(i)$. From node i's perspective, consider two possible scenarios: (a) nodes in $N_i^- \cap F$ are all faulty, and the other nodes are fault-free, and (b) nodes in $N_i^- \cap (C \cup R) = N_i^- \cap F_x(i) \cap \mathcal{V}_F$ are all faulty, and the other nodes are fault-free. Note that, since $F_x(i)$ is a feasible fault set, $N_i^- \cap F_x(i) \cap \mathcal{V}_F$ is also a feasible fault set. Similarly, since F is a feasible fault set, $N_i^- \cap F$ is also a feasible fault set.

  In scenario (a), from node i's perspective, the fault-free nodes have sent values in interval $[m, M]$, whereas the faulty incoming neighbors, i.e., nodes in $N_i^- \cap F$, have sent value m^-. According to the validity condition, $v_i[1] \geq m$. On the other hand, in scenario (b), the fault-free incoming neighbors have sent values m^- and m, where $m^- < m$; so $v_i[1] \leq m$, according to the validity condition. Since node i does not know whether the correct scenario is (a) or (b), it must update its state to satisfy the validity condition in both cases. Thus, it follows that $v_i[1] = m$.

- **At most one of $N_i^- \cap F$ and $N_i^- \cap (C \cup R)$ is non-empty:** Recall that $N_i^- \cap F$ and $N_i^- \cap (C \cup R) = N_i^- \cap F_x(i) \cap \mathcal{V}_F$ are both feasible fault sets. Since at least one of these two sets is empty, their union, i.e., $(N_i^- \cap F) \cup (N_i^- \cap (C \cup R))$, is also a feasible fault set.

Then, from node i's perspective, it is possible that all the nodes in $(N_i^- \cap F) \cup (N_i^- \cap (C \cup R))$ are faulty, and the rest of the nodes are fault-free. In this situation, the values sent to node i by the fault-free nodes (which are all in $\{i\} \cup (N_i^- \cap L)$) are all m, and therefore, $v_i[1]$ must be set to m as per the validity condition.

Hence, $v_i[1] = m$ for each node $i \in L$. Similarly, we can show that $v_j[1] = M$ for each node $j \in R$.

Now consider the nodes in set C (if non-empty). All the values received by the nodes in C are in $[m, M]$, therefore, their new state must also remain in $[m, M]$, as per the validity condition.

The above discussion implies that, at the end of iteration 1, the following conditions hold true: (i) state of each node in L is m , (ii) state of each node in R is M, and (iii) state of each node in C (if non-empty) is in the interval $[m, M]$. These conditions are identical to the initial conditions listed previously. Then, by a repeated application of the above argument (proof by induction), it follows that for any $t \geq 0, v_i[t] = m$ for all nodes $i \in L$, $v_j[t] = M$ for all nodes $j \in R$ and $v_k[t] \in [m, M]$ for all nodes $k \in C$.

Since L and R both contain fault-free nodes, and $m \neq M$, the *convergence* requirement is not satisfied. This is a contradiction to the assumption that a correct iterative algorithm exists in $G(\mathcal{V}, \mathcal{E})$. □

5.4 Algorithm 2

We will prove that there exists an IABC algorithm – particularly *Algorithm 2* below – that satisfies the *validity* and *convergence* conditions provided that the graph $G(\mathcal{V}, \mathcal{E})$ satisfies the necessary condition in Theorem 12. This implies that the necessary condition in Theorem 12 is also sufficient. *Algorithm 2* has the three-step structure described in Chapter 4.2. This algorithm is a generalization – to accommodate the generalized fault model – of iterative algorithms that were analyzed in prior work [29, 52, 41, 50, 49] and Chapter 4. The key difference from previous algorithms is in the *Update* step below.

Algorithm 2

1. *Transmit step:* Transmit current state $v_i[t-1]$ on all outgoing edges and self-loop.

2. Receive step: Receive values on all incoming edges and self-loop. These values form vector $r_i[t]$ of size $|N_i^-| + 1$ (including the value from node i itself). When a fault-free node expects to receive a message from an incoming neighbor but does not receive the message, the message value is assumed to be equal to some *default value*.

3. *Update step:* Sort the values in $r_i[t]$ in an increasing order (breaking ties arbitrarily). Let D be a vector of nodes arranged in an order "consistent" with $r_i[t]$: specifically, $D(1)$ is the node that sent

the smallest value in $r_i[t]$, $D(2)$ is the node that sent the second smallest value in $r_i[t]$, and so on. The size of vector D is also $|N_i^-|+1$.

From vector $r_i[t]$, eliminate the smallest f_1 values, and the largest f_2 values, where f_1 and f_2 are defined as follows:

- f_1 is the largest number such that there exists a feasible fault set $F' \subseteq N_i^-$ containing nodes $D(1), D(2), ..., D(f_1)$. Recall that $i \notin N_i^-$.
- f_2 is the largest number such that there exists a feasible fault set $F'' \subseteq N_i^-$ containing nodes $D(|N_i^-| - f_2 + 2), D(|N_i^-| - f_2 + 3), ..., D(|N_i^-| + 1)$.

F' and F'' above may or may not be identical.

Let $N_i^*[t]$ denote the set of nodes from whom the remaining $|N_i^-| + 1 - f_1 - f_2$ values in $r_i[t]$ were received, and let w_j denote the value received from node $j \in N_i^*[t]$. Note that $i \in N_i^*[t]$. Hence, for convenience, define $w_i = v_i[t-1]$ to be the value node i receives from itself. Observe that if $j \in N_i^*[t]$ is fault-free, then $w_j = v_j[t-1]$.

Define

$$v_i[t] \;=\; Z_i(r_i[t]) \;=\; \sum_{j \in N_i^*[t]} a_i\, w_j \tag{5.1}$$

where

$$a_i = \frac{1}{|N_i^*[t]|} = \frac{1}{|N_i^-| + 1 - f_1 - f_2}$$

The “weight” of each term on the right-hand side of (5.1) is a_i, and these weights add to 1. Also, $0 < a_i \leq 1$. Although f_1, f_2 and a_i may be different for each iteration t, for simplicity, we do not explicitly represent this dependence on t in the notations.

Observe $f_1 + f_2$ nodes whose values are eliminated in the *Update* step above are all in N_i^-. Thus, the above algorithm can be implemented by node i if it knows which of its incoming neighbors may fail simultaneously; node i does not need to know the entire fault domain $\mathcal{F}$ as such.

The main difference between the above algorithm and IABC algorithms in prior work, including Algorithm 1 in Chapter 4.4, is in the choice of the values eliminated from vector $r_i[t]$ in the *Update* step. The manner in which the values are eliminated ensures that the values received from nodes $D(f_1+1)$ and $D(|N_i^-| - f_2 + 1)$ (i.e., the smallest and largest values that survive in $r_i[t]$) are within the convex hull of the state of fault-free nodes, even if nodes $D(f_1+1)$ and $D(|N_i^-|-f_2+1)$ may not be fault-free. This property is useful in proving algorithm correctness (as discussed below).

5.5 Sufficiency (Correctness of Algorithm 2)

We will show that Algorithm 2 satisfies validity and convergence conditions, provided that $G(\mathcal{V},\mathcal{E})$ satisfies the condition below, which matches the necessary condition stated in Theorem 12.

Sufficient condition: *Any reduced graph G_F corresponding to any feasible fault set F contains exactly one* source component.

In the rest of this section, we assume that $G(\mathcal{V},\mathcal{F})$ satisfies the above condition. We first introduce some standard matrix tools to facilitate our proof. Then, we develop a *transition matrix* representation of the *Update* step in Algorithm 2, and show how to use these tools to prove the correctness of Algorithm 2 in $G(\mathcal{V},\mathcal{F})$.

When presenting matrix products, for convenience of presentation, we adopt the "backward" product convention below, where $a \leq b$,

$$\Pi_{i=a}^{b}\mathbf{A}[i] = \mathbf{A}[b]\mathbf{A}[b-1]\cdots\mathbf{A}[a] \tag{5.2}$$

5.5.1 Matrix Preliminaries

In the discussion below, we use boldface upper case letters to denote matrices, rows of matrices, and their elements. For instance, $\mathbf{A}$ denotes a matrix, $\mathbf{A}_i$ denotes the i-th row of matrix $\mathbf{A}$, and $\mathbf{A}_{ij}$ denotes the element at the intersection of the i-th row and the j-th column of matrix $\mathbf{A}$.

Definition 17 *A vector is said to be* stochastic *if all the elements of the vector are* non-negative*, and the elements add up to 1. A matrix is said to be* <u>row</u> *stochastic if each row of the matrix is a stochastic vector.*

For a row stochastic matrix $\mathbf{A}$, coefficients of ergodicity $\delta(\mathbf{A})$ and $\lambda(\mathbf{A})$ are defined as follows [93]:

$$\begin{aligned}\delta(\mathbf{A}) &= \max_j \max_{i_1,i_2} |\mathbf{A}_{i_1 j} - \mathbf{A}_{i_2 j}| \\ \lambda(\mathbf{A}) &= 1 - \min_{i_1,i_2} \sum_j \min(\mathbf{A}_{i_1 j} , \mathbf{A}_{i_2 j})\end{aligned}$$

It is easy to show that $0 \leq \delta(\mathbf{A}) \leq 1$ and $0 \leq \lambda(\mathbf{A}) \leq 1$, and that the rows of $\mathbf{A}$ are all identical if and only if $\delta(\mathbf{A}) = 0$. Also, $\lambda(\mathbf{A}) = 0$ if and only if $\delta(\mathbf{A}) = 0$.

The next result from [34] establishes a relation between the coefficient of ergodicity $\delta(\cdot)$ of a product of row stochastic matrices, and the coefficients of ergodicity $\lambda(\cdot)$ of the individual matrices defining the product.

Lemma 32 *For any p square row stochastic matrices $\mathbf{A}(1), \mathbf{A}(2), \dots \mathbf{A}(p)$,*

$$\delta(\mathbf{A}(p)\mathbf{A}(p-1)\cdots\mathbf{A}(1)) \;\leq\; \Pi_{i=1}^{p}\ \lambda(\mathbf{A}(i)).$$

Lemma 32 is proved in [34]. It implies that if, for all i, $\lambda(\mathbf{A}(i)) \leq 1-\gamma$ for some γ, where $0 < \gamma \leq 1$, then $\delta(\mathbf{A}(p)\mathbf{A}(p-1)\cdots\mathbf{A}(1))$ will approach zero as p approaches ∞. We now define a *scrambling* matrix [34, 93].

Definition 18 *A row stochastic matrix $\mathbf{A}$ is said to be a* scrambling *matrix if*

$$\lambda(\mathbf{A}) < 1$$

The following lemma follows easily from the above definition of $\lambda(\cdot)$.

Lemma 33 *If any column of a row stochastic matrix $\mathbf{A}$ contains only non-zero elements that are all lower bounded by some constant γ, where $0 < \gamma \leq 1$, then $\mathbf{A}$ is a scrambling matrix, and $\lambda(\mathbf{A}) \leq 1-\gamma$.*

5.5.2 Transition Matrix Representation

In our discussion below, $\mathbf{M}[t]$ is a square matrix, $\mathbf{M}_i[t]$ is the i-th row of the matrix, and $\mathbf{M}_{ij}[t]$ is the element at the intersection of the i-th row and j-th column of $\mathbf{M}[t]$.

For a given execution of Algorithm 2, let F denote the actual set of faulty nodes in that execution. Let $|F| = \psi$. Without loss of generality, suppose that nodes 1 through $(n-\psi)$ are fault-free, and if $\psi > 0$, nodes $(n-\psi+1)$ through n are faulty. Denote by $v[0]$ the column vector consisting of the initial states of all the fault-free nodes. Denote by $v[t]$, where $t \geq 1$, the column vector consisting of the states of all the fault-free nodes at the end of the t-th iteration. The i-th element of vector $v[t]$ is state $v_i[t]$. The size of vector $v[t]$ is $(n-\psi)$.

We will show that the iterative update of the state of a fault-free node i $(1 \leq i \leq n-\psi)$ performed in (5.1) in Algorithm 2 can be expressed using the matrix form below.

$$v_i[t] = \mathbf{M}_i[t]\ v[t-1] \tag{5.3}$$

where $\mathbf{M}_i[t]$ is a *stochastic row* vector of size $n-\psi$. That is, $\mathbf{M}_{ij}[t] \geq 0$, for $1 \leq j \leq n-\psi$, and $\sum_{1\leq j\leq n-\psi} \mathbf{M}_{ij}[t] = 1$.[2] By "stacking" (5.3) for different i, $1 \leq i \leq n-\psi$, we will represent the *Update* step of Algorithm 2 at all the fault-free nodes together using (5.4) below.

$$v[t] = \mathbf{M}[t]\ v[t-1] \tag{5.4}$$

where $\mathbf{M}[t]$ is a $(n-\psi)\times(n-\psi)$ *row stochastic* matrix, with its i-th row being equal to $\mathbf{M}_i[t]$ in (5.3). $\mathbf{M}[t]$ is said to be a transition matrix.

By repeated application of (5.4), we can represent the *Update* step of Algorithm 2 at the t-th iterations ($t \geq 1$) as:

$$v[t] \quad = \quad \left(\Pi_{k=1}^{t}\mathbf{M}[k]\right)\ v[0] \tag{5.5}$$

Recall that we adopt the "backward" product convention as presented in (5.2).

In the rest of this section, we will first "construct" transition matrices $\mathbf{M}[k]$ ($1 \leq k \leq t$) that satisfy certain desirable properties. Then, we will identify a connection between these transition matrices and the sufficiency condition stated above, and use this connection to establish *convergence* property for Algorithm 2. The *validity* property also follows from the transition matrix representation.

5.5.3 Construction of Transition Matrix

We will construct a transition matrix with the property described in Lemma 34 below.

Lemma 34 *The* Update step *of Algorithm 2 at the fault-free nodes can be expressed using row stochastic transition matrix* $\mathbf{M}[t]$*, such that there exists a feasible fault set* $F_x(i)$ *for each* $i \in \mathcal{V}-F$ *such that, for all* $j \in \{i\} \cup ((\mathcal{V}_F - F_x(i)) \cap N_i^-)$,

$$\mathbf{M}_{ij}[t] \ \geq \ \beta$$

where β *is a constant (to be defined later), and* $0 < \beta \leq 1$.

Proof: We prove the correctness of Lemma 34 by constructing $\mathbf{M}_i[t]$ for $1 \leq i \leq n-\psi$ that satisfies the conditions in Lemma 34. Recall that F is the set of faulty nodes, and $|F| = \psi$. As stated before, without loss of generality, nodes 1 through $n-\psi$ are assumed to be fault-free, and the remaining ψ nodes faulty.

[2] In addition to t, the row vector $\mathbf{M}_i[t]$ may depend on the state vector $v[t-1]$ as well as the behavior of the faulty nodes in F. For simplicity, the notation $\mathbf{M}_i[t]$ does not explicitly represent this dependence.

Consider a fault-free node i performing the *Update* step in Algorithm 2. In the *Update* step, recall that the smallest f_1 and the largest f_2 values are eliminated from $r_i[t]$, where the choice of f_1 and f_2 is described in Algorithm 2. Let us denote by $\mathcal{S}$ and $\mathcal{L}$, respectively, the set of nodes[3] from whom the smallest f_1 and the largest f_2 values were received by node i in iteration t. Define sets $\mathcal{S}_g$ and $\mathcal{L}_g$ to be subsets of $\mathcal{S}$ and $\mathcal{L}$ that contain all the fault-free nodes in $\mathcal{S}$ and $\mathcal{L}$, respectively. That is, $\mathcal{S}_g = \mathcal{S} \cap (\mathcal{V} - F)$ and $\mathcal{L}_g = \mathcal{L} \cap (\mathcal{V} - F)$.

Construction of $\mathbf{M}_i[t]$ differs somewhat depending on whether sets $\mathcal{S}_g, \mathcal{L}_g$ and $N_i^*[t] \cap F$ are empty or non-empty. We divide the possibilities into 6 separate cases:

- Case I: $\mathcal{S}_g \neq \Phi, \mathcal{L}_g \neq \Phi$, and $N_i^*[t] \cap F \neq \Phi$.
- Case II: $\mathcal{S}_g \neq \Phi, \mathcal{L}_g \neq \Phi$, and $N_i^*[t] \cap F = \Phi$.
- Case III: $\mathcal{S}_g = \Phi, \mathcal{L}_g \neq \Phi$, and $N_i^*[t] \cap F \neq \Phi$.
- Case IV: $\mathcal{S}_g \neq \Phi, \mathcal{L}_g = \Phi$, and $N_i^*[t] \cap F \neq \Phi$.
- Case V: $\mathcal{S}_g = \Phi, \mathcal{L}_g = \Phi$, and $N_i^*[t] \cap F \neq \Phi$.
- Case VI: at most one of $\mathcal{S}_g$ and $\mathcal{L}_g$ is non-empty, and $N_i^*[t] \cap F = \Phi$.

Case I In Case I, $\mathcal{S}_g \neq \Phi, \mathcal{L}_g \neq \Phi$, and $N_i^*[t] \cap F \neq \Phi$. Let $m_{\mathcal{S}}$ and $m_{\mathcal{L}}$ be defined as shown below. Recall that the nodes in $\mathcal{S}_g$ and $\mathcal{L}_g$ are all fault-free, and therefore, for any node $j \in \mathcal{S}_g \cup \mathcal{L}_g$, $w_j = v_j[t-1]$ (in the notation of Algorithm 2).

$$m_{\mathcal{S}} = \frac{\sum_{j \in \mathcal{S}_g} v_j[t-1]}{|\mathcal{S}_g|} \qquad \text{and} \qquad m_{\mathcal{L}} = \frac{\sum_{j \in \mathcal{L}_g} v_j[t-1]}{|\mathcal{L}_g|}$$

Now, consider any node $k \in N_i^*[t]$. By the definition of sets $\mathcal{S}_g$ and $\mathcal{L}_g$, $m_{\mathcal{S}} \leq w_k \leq m_{\mathcal{L}}$. Therefore, we can find weights $S_k \geq 0$ and $L_k \geq 0$ such that $S_k + L_k = 1$, and

$$\begin{aligned} w_k &= S_k \; m_{\mathcal{S}} + L_k \; m_{\mathcal{L}} && (5.6) \\ &= \frac{S_k}{|\mathcal{S}_g|} \sum_{j \in \mathcal{S}_g} v_j[t-1] + \frac{L_k}{|\mathcal{L}_g|} \sum_{j \in \mathcal{L}_g} v_j[t-1] && (5.7) \end{aligned}$$

Clearly, at least one of S_k and L_k must be $\geq 1/2$. We now define elements $\mathbf{M}_{ij}[t]$ of row $\mathbf{M}_i[t]$:

[3] Although $\mathcal{S}$ and $\mathcal{L}$ may be different for each t, for simplicity, we do not explicitly represent this dependence on t in the notations $\mathcal{S}$ and $\mathcal{L}$.

- For $j \in N_i^*[t] \cap (\mathcal{V} - F)$: In this case, j is either a fault-free incoming neighbor of i, or i itself. For each such j, define $\mathbf{M}_{ij}[t] = a_i$. This is obtained by observing in (5.1) that the contribution of such a node j to the new state $v_i[t]$ is $a_i \; w_j = a_i \; v_j[t-1]$.

  The elements of $\mathbf{M}_i[t]$ defined here add up to

$$|N_i^*[t] \cap (\mathcal{V} - F)| \; a_i$$

- For $j \in \mathcal{S}_g \cup \mathcal{L}_g$: In this case, j is a fault-free node in $\mathcal{S}$ or $\mathcal{L}$.

  For each $j \in \mathcal{S}_g$,

$$\mathbf{M}_{ij}[t] \;=\; a_i \sum_{k \in N_i^*[t] \cap F} \frac{S_k}{|\mathcal{S}_g|}$$

  and for each node $j \in \mathcal{L}_g$,

$$\mathbf{M}_{ij}[t] \;=\; a_i \sum_{k \in N_i^*[t] \cap F} \frac{L_k}{|\mathcal{L}_g|}$$

  To obtain these two expressions, we represent value w_k sent by each faulty node k in $N_i^*[t]$, i.e., $k \in N_i^*[t] \cap F$, using (5.7). Recall that this node k contributes $a_i w_k$ to (5.1). The above two expressions are then obtained by summing (5.7) over all the faulty nodes in $N_i^*[t] \cap F$, and replacing this sum by equivalent contributions by nodes in $\mathcal{S}_g$ and $\mathcal{L}_g$.

  The elements of $\mathbf{M}_i[t]$ defined here add up to

$$a_i \sum_{k \in N_i^*[t] \cap F} (S_k + L_k) = |N_i^*[t] \cap F| \; a_i.$$

- For $j \in (\mathcal{V} - F) - (N_i^*[t] \cup \mathcal{S}_g \cup \mathcal{L}_g)$: These fault-free nodes have not yet been considered above. For each such node j, define $\mathbf{M}_{ij}[t] = 0$.

With the above definition of $\mathbf{M}_i[t]$, it should be easy to see that $\mathbf{M}_i[t] \, v[t-1]$ is, in fact, identical to $v_i[t]$ obtained using (5.1). Thus, the above construction of $\mathbf{M}_i[t]$ results in the contribution of the faulty nodes in $N_i^*[t]$ to (5.1) being replaced by an equivalent contribution from fault-free nodes in $\mathcal{L}_g$ and $\mathcal{S}_g$.

Properties of $\boldsymbol{M}_i[t]$:

First, we show that $\mathbf{M}[t]$ is row stochastic. Observe that all the elements of $\mathbf{M}_i[t]$ are non-negative. Also, all the elements of $\mathbf{M}_i[t]$ above add up to

$$|N_i^*[t] \cap (\mathcal{V} - F)| \; a_i + |N_i^*[t] \cap F| \; a_i = |N_i^*[t]| \; a_i = 1$$

because $a_i = 1/|N_i^*[t]|$ as defined in Algorithm 2. Thus, $\mathbf{M}_i[t]$ is a stochastic row vector.

Recall that from the above discussion, for $k \in N_i^*[t]$, one of S_k and L_k must be $\geq 1/2$. Without loss of generality, assume that $S_s \geq 1/2$ for some $s \in N_i^*[t] \cap F$. Consequently, for each node $j \in \mathcal{S}_g$, $\mathbf{M}_{ij}[t] \geq \frac{a_i}{|\mathcal{S}_g|} S_s \geq \frac{a_i}{2|\mathcal{S}_g|}$. Also, for each fault-free node j in $N_i^*[t]$, $\mathbf{M}_{ij}[t] = a_i$. Thus, if β is chosen such that

$$0 < \beta \leq \frac{a_i}{2|\mathcal{S}_g|} \tag{5.8}$$

and $F_x(i)$ is defined to be equal to $\mathcal{L}$, then the condition in the lemma holds for node i. That is, $\mathbf{M}_{ij}[t] \geq \beta$ for $j \in \{i\} \cup ((\mathcal{V}_F - F_x(i)) \cap N_i^-)$.

Case II Now, we consider the case when $\mathcal{S}_g \neq \Phi, \mathcal{L}_g \neq \Phi$, and $N_i^*[t] \cap F = \Phi$. That is, when each of $\mathcal{S}$ and $\mathcal{L}$ contains at least one fault-free node, and $N_i^*[t]$ contains only fault-free node(s). In fact, the analysis for Case II is very similar to the one for Case I when $N_i^*[t]$ does contain a faulty node. We now discuss how the analysis of Case I can be applied to Case II. Rewrite (5.1) as follows:

$$\begin{aligned} v_i[t] &= \frac{a_i}{2} v_i[t-1] + \frac{a_i}{2} v_i[t-1] + \sum_{j \in N_i^*[t] - \{i\}} a_i w_j & (5.9) \\ &= a_i w_z + a_i w_i + \sum_{j \in N_i^*[t] - \{i\}} a_i w_j & (5.10) \end{aligned}$$

In the above equation, z is to be viewed as a "virtual" incoming neighbor of node i, which has sent value $w_z = \frac{v_i[t-1]}{2}$ to node i in iteration t. With the above rewriting of state update, the value received by node i from itself should be viewed as $w_i = \frac{v_i[t-1]}{2}$ instead of $v_i[t-1]$. With this transformation, Case II now becomes identical to Case I, with virtual node z being treated as an incoming neighbor of node i.

In essence, a part of node i's contribution (half, to be precise) is now replaced by equivalent contribution by nodes in $\mathcal{L}_g$ and $\mathcal{S}_g$. We now define elements $\mathbf{M}_{ij}[t]$ of row $\mathbf{M}_i[t]$:

- For $j = i$: $\mathbf{M}_{ij}[t] = \frac{a_i}{2}$. This is obtained by observing in (5.1) that node i's contribution to the new state $v_i[t]$ is $a_i \frac{v_i[t-1]}{2}$.
- For $j \in N_i^*[t] - \{i\}$: In this case, j is a fault-free incoming neighbor of i. For each such j, define $\mathbf{M}_{ij}[t] = a_i$. This is obtained by observing in (5.1) that the contribution of node j to the new state $v_i[t]$ is $a_i w_j = a_i v_j[t-1]$.
- For $j \in \mathcal{S}_g \cup \mathcal{L}_g$: In this case, j is a fault-free node in $\mathcal{S}$ or $\mathcal{L}$.

For each $j \in \mathcal{S}_g$,

$$\mathbf{M}_{ij}[t] \;=\; \frac{a_i}{2}\frac{S_z}{|\mathcal{S}_g|}$$

and for each node $j \in \mathcal{L}_g$,

$$\mathbf{M}_{ij}[t] \;=\; \frac{a_i}{2}\frac{L_z}{|\mathcal{L}_g|}$$

where S_z and L_z are chosen such that $S_z + L_z = 1$ and $w_z = \frac{v_i[t-1]}{2} = \frac{S_z}{2}m_{\mathcal{S}} + \frac{L_z}{2}m_{\mathcal{L}}$. Note that such S_z and L_z exist because by definition of $\mathcal{S}_g$ and $\mathcal{L}_g$, $v_i[t-1] \geq w_j,\ \forall j \in S_g$ and $v_i[t-1] \leq w_j,\ \forall j \in L_g$. Then the two expressions above are obtained by replacing the contribution of the virtual node z by an equivalent contribution by the nodes in $\mathcal{S}_g$ and $\mathcal{L}_g$, respectively.

- For $j \in (\mathcal{V} - F) - (N_i^*[t] \cup \mathcal{S}_g \cup \mathcal{L}_g)$: These fault-free nodes have not yet been considered above. For each such node j, define $\mathbf{M}_{ij}[t] = 0$.

Properties of $\boldsymbol{M}_i[t]$:

By argument similar to the one in Case I, $\mathbf{M}[t]$ is row stochastic. Without loss of generality, suppose that $S_z \geq 1/2$. Then for each node $j \in \mathcal{S}_g$, $\mathbf{M}_{ij}[t] = \frac{a_i}{2|\mathcal{S}_g|}S_z \geq \frac{a_i}{4|\mathcal{S}_g|}$. Also, for fault-free node j in $N_i^*[t] - \{i\}$, $\mathbf{M}_{ij}[t] = a_i$, and $\mathbf{M}_{ii}[t] = \frac{a_i}{2}$. Recall that by definition, $|\mathcal{S}_g| \geq 1$. Hence, if β is chosen such that

$$0 < \beta \leq \frac{a_i}{4|\mathcal{S}_g|} \tag{5.11}$$

and $F_x(i)$ is defined to be equal to $\mathcal{L}$, then the condition in the Lemma 34 holds for node i. That is, $\mathbf{M}_{ij}[t] \geq \beta$ for $j \in \{i\} \cup (\mathcal{V}_F - F_x(i)) \cap N_i^-$.

Cases III and IV Now, we describe the construction of Case III. The construction for Case IV is very similar, and thus, is omitted here.

In Case III, $\mathcal{S}_g = \Phi, \mathcal{L}_g \neq \Phi$, and $N_i^*[t] \cap F \neq \Phi$. Thus, $\mathcal{S}$ does not contain any fault-free nodes (hence $\mathcal{S}_g$ is empty). This may be due to one of the following two reasons: (i) the set $\mathcal{S}$ is non-empty, but all the nodes in $\mathcal{S}$ are faulty, or (ii) set $\mathcal{S}$ is empty.

Assume that $l \in \mathcal{L}$ is a fault-free node, and that all the nodes in $\mathcal{S}$ are faulty (i.e., $\mathcal{S}_g = \Phi$) or that $\mathcal{S}$ is empty (i.e., $f_1 = 0$). In this case, observe that node $D(f_1 + 1)$ must be fault-free (otherwise, f_1 cannot be the largest value as defined in Algorithm 2). Now, consider any node $k \in N_i^*[t]$. Similar to the argument in

Case I, we can find weights $S_k \geq 0$ and $L_k \geq 0$ such that

$$S_k + L_k = 1$$

and

$$w_k = S_k \ v_{D(f_1+1)}[t-1] + L_k \ v_l[t-1] \tag{5.12}$$

We now define $\mathbf{M}_{ij}[t]$ for all fault-free j.

- For $j \in (N_i^*[t] - \{D(f_1+1)\}) \cap (\mathcal{V} - F)$. That is, j is a fault-free node in $N_i^*[t]$ with the exception of $D(f_1+1)$.

  For each such j, define $\mathbf{M}_{ij}[t] = a_i$. This is obtained by observing in (5.1) that the contribution of node j to the new state $v_i[t]$ is $a_i w_j = a_i \ v_j[t-1]$.

  The elements of $\mathbf{M}_i[t]$ defined here (including the case of $j = i$) add up to

  $$(|N_i^*[t] \cap (\mathcal{V} - F)| - 1) \ a_i.$$

- For nodes $D(f_1+1)$ and l: Define

  $$\mathbf{M}_{iD(f_1+1)}[t] = a_i + \sum_{k \in N_i^*[t] \cap F} a_i \ S_k$$

  and

  $$\mathbf{M}_{il}[t] = \sum_{k \in N_i^*[t] \cap F} a_i \ L_k$$

  Similar to Case I, these two expressions are obtained by summing up the contribution over the faulty nodes in $N_i^*[t]$, and replacing the sum by an equivalent contribution by the nodes $D(f_1+1)$ and l, respectively, according to (5.12).

  The above elements of $\mathbf{M}_i[t]$ add up to

  $$a_i \left(1 + \sum_{k \in N_i^*[t] \cap F} (S_k + L_k) \right) = (1 + |N_i^*[t] \cap F|) \ a_i.$$

- For $j \in (\mathcal{V} - F) - (N_i^*[t] \cup \{l\})$: These fault-free nodes have not yet been considered above. For each such j, define $M_{ij}[t] = 0$.

Similar to Case I, in Case III as well, it should be easy to see that

$$\mathbf{M}_i[t]\, v[t-1]$$

is identical to $v_i[t]$ obtained using (5.1).

Properties of $\boldsymbol{M}_i[t]$: All the elements of $\mathbf{M}_i[t]$ are non-negative. The elements of $\mathbf{M}_i[t]$ defined in Case II add up to

$$(|N_i^*[t] \cap (\mathcal{V} - F)| - 1)\ a_i + (1 + |N_i^*[t] \cap F|)\ a_i = |N_i^*[t]|\ a_i = 1$$

Thus, $\mathbf{M}_i[t]$ is a stochastic row vector.

In Case III, recall that for any fault-free node j in $N_i^*[t]$ (including $j = D(f_1+1)$ and $j = i$), $\mathbf{M}_{ij}[t] \geq a_i$. Thus, if β is chosen such that

$$0 < \beta \leq a_i \tag{5.13}$$

and $F_x(i)$ is defined to be equal to $\mathcal{L}$, then the condition in the Lemma 34 holds for node i.

Case V Consider Case V, where $N_i^*[t] \cap F \neq \Phi$, and $\mathcal{S}_g = \mathcal{L}_g = \Phi$. In this case, it should be easy to see that $N_i^*[t]$ contains at least 3 nodes. In particular, D_{f_1+1} must be fault-free (otherwise, f_1 cannot be maximum possible), $D_{|N_i^-|-f_2+1}$ must be fault-free (otherwise, f_2 cannot be maximum possible), and there is a faulty node in $N_i^*[t]$.

Now this case can be handled similar to Case III analyzed above. In particular, entries in $\mathbf{M}_i[t]$ are defined similarly with l being defined equal to $D_{N_i^- - f_2+1}$. Also, define $F_x(i) = \Phi$.

Hence, it is easy to see that the properties of $\mathbf{M}_i[t]$ are identical to Case III presented above.

Case VI Here, we consider the case when at most one of $\mathcal{S}$ and $\mathcal{L}$ contains a fault-free node and $N_i^*[t] \cap F = \Phi$. Without loss of generality, suppose that $\mathcal{S}$ contains only faulty nodes, and $\mathcal{L}$ may contain a fault-free node.

In this case, define $\mathbf{M}_{ij}[t] = a_i$ for $j \in N_i^*[t]$; define $\mathbf{M}_{ij} = 0$ for all other fault-free nodes j. Also, define $F_x(i) = \mathcal{L}$.

The properties of $\mathbf{M}_i[t]$ thus defined are identical to Case III above.

All Cases Together Now, let us consider Cases I-VI together. From the definition of a_i in Algorithm 2, observe that $a_i \geq \frac{1}{|N_i^-|+1}$ (because $f_1, f_2 \geq 0$). Let us define

$$\alpha = \min_{i \in \mathcal{V}} \frac{1}{|N_i^-|+1}$$

Moreover, observe that $|\mathcal{S}_g| \leq n$ and $|\mathcal{L}_g| \leq n$. Then define β as

$$\beta = \frac{\alpha}{4n} \tag{5.14}$$

This definition satisfies constraints on β in Cases I through VI (conditions (5.8), (5.11) and (5.13)). Thus, Lemma 34 holds for all six cases with this choice of β. □

5.5.4 Validity and Convergence of Algorithm 2

Now, we are ready to prove that Algorithm 2 is correct. We first discuss the properties of $\mathbf{M}[t]$, and use the properties in the correctness proof.

Correspondence between $\mathbf{M}[t]$ and a Reduced Graph Let R_F denote the set of all the reduced graphs of $G(\mathcal{V}, \mathcal{E})$ corresponding to a feasible fault set F. Let $\tau = |R_F|$. τ depends on F and the underlying network, and is finite.

In this discussion, let us denote a reduced graph by an italic upper case letter, and the corresponding "connectivity matrix" (defined below) using the same letter in boldface upper case. Thus, $\mathbf{H}$ denotes the connectivity matrix for graph $H \in R_F$.

Non-zero elements of connectivity matrix $\mathbf{H}$ are defined as follows: (i) for $1 \leq i, j \leq n - \psi$, $\mathbf{H}_{ij} = 1$ if and only if $(j, i) \in H$, and (ii) $\mathbf{H}_{ii} = 1$ for $1 \leq i \leq n - \psi$. That is, non-zero elements of row $\mathbf{H}_i$ correspond to the incoming links at node i, and the self-loop at node i. Thus, the connectivity matrix for any reduced graph in R_F has a non-zero diagonal.

Based on the *sufficient condition* stated at the start of Chapter 5.5 and Lemma 34, we can show the following key lemmas.

Lemma 35 *For any $H \in R_F, \mathbf{H}^{\mathbf{n}-\psi}$ has at least one non-zero column.*

Proof: $G(\mathcal{V}, \mathcal{E})$ satisfies the *sufficient condition* stated at the start of Chapter 5.5. Therefore, there exists at least one non-faulty node k in the reduced graph H that has directed paths to all the nodes in H (consisting of the edges in H). Since the length of the path from k to any other node in H is at most $n-\psi-1$, the k-th column of matrix $\mathbf{H}^{n-\psi}$ will be non-zero.[4] □

Definition 19 *For matrices $\boldsymbol{A}$ and $\boldsymbol{B}$ of identical size, and a scalar γ, $\gamma\boldsymbol{B} \le \boldsymbol{A}$ provided that $\gamma\boldsymbol{B}_{ij} \le \boldsymbol{A}_{ij}$ for all i, j.*

Lemma 36 *For any $t \ge 1$, there exists a graph $H \in R_F$ such that $\beta\mathbf{H} \le \mathbf{M}[t]$.*

Proof: Observe that the i-th row of the transition matrix $\mathbf{M}[t]$ corresponds to the state update (in Algorithm 2) performed at fault-free node i. Recall from Lemma 34 that $\mathbf{M}_{ij}[t] \ge \beta$ for $j \in \{i\} \cup ((\mathcal{V}_F - F_x(i)) \cap N_i^-)$, where $F_x(i)$ is a feasible fault set.

Let us obtain a reduced graph H by choosing $F_x(i)$ for each i as defined in Lemma 34. Then from the definition of connectivity matrix $\mathbf{H}$, Lemma 36 then follows. □

Correctness of Algorithm 2 The rest of the proof below is inspired by related work on non-fault-tolerant consensus [38].

Let $\mathbf{H}[t]$ denote the matrix $\mathbf{H}$ corresponding to $\mathbf{M}[t]$ as defined in Lemma 36.

Lemma 37 *For any $z \ge 1$, in the product below of $\mathbf{H}[t]$ matrices for consecutive $\tau(n-\psi)$ iterations, at least one column is non-zero.*

$$\Pi_{t=z}^{z+\tau(n-\psi)-1}\,\mathbf{H}[t]$$

Proof: Since the above product consists of $\tau(n-\psi)$ connectivity matrices corresponding to graphs in $R_{\mathcal{F}}$, at least one of the connectivity matrices corresponding to the τ distinct graphs in $R_{\mathcal{F}}$, say matrix $\mathbf{H}_*$, will appear in the above product at least $n-\psi$ times.

Now observe that: (i) By Lemma 35, $\mathbf{H}_*^{n-\psi}$ contains a non-zero column, say the k-th column is non-zero, and (ii) all the $\mathbf{H}[t]$ matrices in the product contain a non-zero diagonal. These two observations together imply that the k-th column in the above product is non-zero. □

Let us now define a sequence of matrices $\mathbf{Q}(i)$, $i \ge 1$, such that each of these matrices is a product of $\tau(n-\psi)$ of the $\mathbf{M}[t]$ matrices. Specifically,

$$\mathbf{Q}(i) \;=\; \Pi_{t=(i-1)\tau(n-\psi)+1}^{i\tau(n-\psi)}\,\mathbf{M}[t] \tag{5.15}$$

[4] That is, all the elements of the column will be non-zero. Also, such a non-zero column will exist in $\mathbf{H}^{n-\psi-1}$, too. We use the loose bound of $n-\psi$ to simplify the presentation.

From (5.5) and (5.15) observe that

$$v[k\tau(n-\psi)] \quad = \quad \left(\Pi_{i=1}^{k}\, \mathbf{Q}(i)\right)\; v[0] \tag{5.16}$$

Lemma 38 *For $i \geq 1$, $\mathbf{Q}(i)$ is a scrambling row stochastic matrix, and*

$$\lambda(\mathbf{Q}(i)) \leq 1 - \beta^{\tau(n-\psi)}.$$

Proof:

$\mathbf{Q}(i)$ is a product of row stochastic matrices ($\mathbf{M}[t]$); therefore, $\mathbf{Q}(i)$ is row stochastic. From Lemma 36, for each $t \geq 1$,

$$\beta\, \mathbf{H}[t] \;\leq\; \mathbf{M}[t]$$

Therefore,

$$\beta^{\tau(n-\psi)}\; \Pi_{t=(i-1)\tau(n-\psi)+1}^{i\tau(n-\psi)}\, \mathbf{H}[t] \;\leq\; \Pi_{t=(i-1)\tau(n-\psi)+1}^{i\tau(n-\psi)}\, \mathbf{M}[t] \;=\; \mathbf{Q}(i)$$

By using $z = (i-1)(n-\psi)+1$ in Lemma 37, we conclude that the matrix product on the left side of the above inequality contains a non-zero column. Therefore, $\mathbf{Q}(i)$ on the right side of the inequality also contains a non-zero column.

Observe that $\tau(n-\psi)$ is finite, and hence, $\beta^{\tau(n-\psi)}$ is non-zero. Since the non-zero terms in $\mathbf{H}[t]$ matrices are all 1, the non-zero elements in $\Pi_{t=(i-1)\tau(n-\psi)+1}^{i\tau(n-\psi)}\mathbf{H}[t]$ must each be ≥ 1. Therefore, there exists a non-zero column in $\mathbf{Q}(i)$ with all the elements in the column being $\geq \beta^{\tau(n-\psi)}$. Therefore, by Lemma 33, $\lambda(\mathbf{Q}(i)) \leq 1 - \beta^{\tau(n-\psi)}$, and $\mathbf{Q}(i)$ is a scrambling matrix. □

Theorem 13 *Suppose that $G(\mathcal{V}, \mathcal{E})$ satisfies the sufficient condition stated above. Algorithm 2 satisfies both the validity and convergence conditions.*

Proof:

Since $v[t] = \mathbf{M}[t]\, v[t-1]$, and $\mathbf{M}[t]$ is a row stochastic matrix, it follows that Algorithm 2 satisfies the validity condition.

Using Lemma 32 and the definition of $\mathbf{Q}(i)$, and using the inequalities $\lambda(\mathbf{M}[t]) \leq 1$ and $\lambda(\mathbf{Q}(i)) \leq$

$(1 - \beta^{\tau(n-\psi)}) < 1$, we get

$$\begin{aligned}\lim_{t\to\infty} \delta(\Pi_{i=1}^{t}\mathbf{M}[i]) \;&=\; \lim_{t\to\infty} \delta\left(\left(\Pi_{i=(\lfloor \frac{t}{\tau(n-\psi)} \rfloor)\tau(n-\psi)+1}^{t}\mathbf{M}[i]\right)\left(\Pi_{i=1}^{\lfloor \frac{t}{\tau(n-\psi)} \rfloor}\mathbf{Q}(i)\right)\right) \\ &\leq\; \lim_{t\to\infty} \Pi_{i=1}^{\lfloor \frac{t}{\tau(n-\psi)} \rfloor}\lambda(\mathbf{Q}(i)) \;=\; 0\end{aligned}$$

Thus, the rows of $\Pi_{i=1}^{t}\mathbf{M}[i]$ become identical in the limit. This observation, and the fact that $v[t] = (\Pi_{i=1}^{t}\mathbf{M}[i])v[0]$ together imply that the states of the fault-free nodes satisfy the convergence condition. □

5.6 Summary

This Chapter considers a *generalized* fault model [39, 43], which can be used to specify more complex failure patterns, such as correlated failures or non-uniform node reliabilities. Under this fault model, we prove a *tight* necessary condition for the existence of synchronous iterative approximate Byzantine consensus algorithms in arbitrary directed graphs. Then, we show the condition is also sufficient by providing a new IABC algorithm.

We present a *transition matrix*-based proof to show the correctness of the proposed algorithm. While transition matrices have been used to prove correctness of *non-fault tolerant* consensus [38], this Chapter is the first to extend the technique to Byzantine consensus. We also extend this technique to solve other consensus problems in next two Chapters.

Chapter 6

Iterative Approximate Byzantine Consensus Under Link Faults

6.1 Introduction

Previous Chapters consider consensus problems when nodes may be faulty and all links are reliable. This Chapter explores the problem in synchronous systems in which all nodes are fault-free and the links may be Byzantine faulty. This type of fault model has been addressed in prior work [14, 64, 65, 67]. Particularly, we adopt the *transient Byzantine link* failure model [64, 65], in which different sets of link failures may occur at different times (formal definition in Chapter 6.2). We consider the problem in arbitrary directed graphs using iterative algorithms that maintain only a small amount of memory across iterations. The *iterative approximate Byzantine consensus* (IABC) algorithms of interest were discussed in Chapter 4.2 and have the following properties:

- *Initial state* of each node is equal to a real-valued *input* provided to that node.
- *Termination*: The algorithm terminates in finite number of iterations.
- *Validity*: After each iteration of the algorithm, the state of each node must stay in the *convex hull* of the states of all the nodes at the end of the *previous* iteration.
- *ϵ-agreement*: For any $\epsilon > 0$, when the algorithm terminates, the difference between the outputs at any pair of nodes is guaranteed to be within ϵ.

Note that in Chapters 4 and 5, we stated the last property a bit differently, namely, convergence property. Effectively, ϵ-agreement provides a precise termination condition so that the convergence property is satisfied.

This Chapter extends our work on approximate consensus under node failures presented in Chapter 4 and 5. We identify a *tight* necessary and sufficient condition for the graphs to be able to reach approximate consensus under *transient Byzantine link* failure models [64, 65] using restricted iterative algorithms; our proof of correctness follows the same proof structure presented in Chapter 5 and [87, 86]. The use of matrix analysis is inspired by the prior work on non-fault-tolerant consensus (e.g., [38, 9, 34]). The results presented in this Chapter are published in [81].

6.2 Transient Byzantine Link Fault Model

Fault Model The system is assumed to be synchronous. And we consider the transient Byzantine *link* failure model [64, 65] for iterative algorithms in directed network. All nodes are assumed to be *fault-free*, and only send a single message on each outgoing edge in each iteration. A link (i, j) is said to be faulty *in a certain iteration* if the message sent by node i is different from the message received by node j in that iteration, i.e., the message from i to j is corrupted. Note that in our model, it is possible that link (i, j) is faulty while link (j, i) is fault-free.[1] In every iteration, up to f links may be faulty, i.e., at most f links may deliver corrupted messages or drop messages. Note that different sets of link failures may occur in different iterations.

A faulty link may tamper or drop messages. Also, the faulty links may be controlled by a single omniscient adversary. The adversary is assumed to have a complete knowledge of the execution of the algorithm, including the states of all the nodes, contents of the messages exchanged, the algorithm specification, and the network topology.

6.3 Necessary Condition

For a correct iterative approximate consensus algorithm to exist under transient Byzantine link failures, the graph $G = (\mathcal{V}, \mathcal{E})$ must satisfy the necessary condition proved in this section. Recall that in Chapter 4.3, we have introduced relations $\Rightarrow$ and $\not\Rightarrow$ in Definition 10. This Chapter will use these relations to define the tight condition. The condition is similar to prior conditions – except that instead of isolating a set of f nodes, we remove a set of f links to accommodate potential misbehaviors by faulty links.

Condition P : Consider graph $G = (\mathcal{V}, \mathcal{E})$. Denote by F a subset of $\mathcal{E}$ such that $|F| \leq f$. Let sets L, C, R form a partition of $\mathcal{V}$, such that both L and R are non-empty. Then, in $G' = (\mathcal{V}, \mathcal{E} - F)$, at least one of the two conditions below must be true: (i) $C \cup R \Rightarrow L$ or (ii) $L \cup C \Rightarrow R$.

The necessity proof below bears some similarity in the necessity proof of Theorem 6. One difference is the way modeling link failures instead of node failures. We present the proof here for completeness.

Theorem 14 *Suppose that a correct IABC algorithm exists for $G = (\mathcal{V}, \mathcal{E})$. Then G satisfies* Condition P.

Proof:

[1] For example, the described case is possible in wireless network, if node i's transmitter is broken while node i's receiver and node j's transmitter and receiver all function correctly.

The proof is by contradiction. Let us assume that a correct IABC algorithm exists in $G = (\mathcal{V}, \mathcal{E})$, and for some node partition L, C, R of $\mathcal{V}$ and a subset $F \subseteq \mathcal{E}$ such that $|F| \leq f$, $C \cup R \not\Rightarrow L$ and $L \cup C \not\Rightarrow R$ in $G' = (\mathcal{V}, \mathcal{E}')$, where $\mathcal{E}' = \mathcal{E} - F$. Thus, for any $i \in L$, $|\{(k,i) \mid k \in C \cup R,\ (k,i) \in \mathcal{E} - F\}| \leq f$. Similarly, for any $j \in R$, $|\{(k,j) \mid k \in L \cup C,\ (k,j) \in \mathcal{E} - F\}| \leq f$.

Also assume that all the links in F (if F is non-empty) are faulty, and the rest of the links are fault-free in every iteration. Note that the nodes are not aware of the identity of the faulty links.

Consider the case when (i) each node in L has initial input m, (ii) each node in R has initial input M, such that $M > m + \epsilon$, and (iii) each node in C, if C is non-empty, has an input in the interval $[m, M]$. Define m^- and M^+ such that $m^- < m$ and $M < M^+$.

In the *Transmit Step* of iteration 1 in the IABC algorithm, each node k, sends to nodes in N_k^+ value $v_k[0]$; however, some values sent via faulty links may be tampered. Suppose that the messages sent via the faulty links in F (if non-empty) are tampered in the following way: (i) if the link is an incoming link to a node in L, then $m^- < m$ is delivered to that node; (ii) if the link is an incoming link to a node in R, then $M^+ > M$ is delivered to that node; and (iii) if the link is an incoming link to a node in C, then some arbitrary value in interval $[m, M]$ is delivered to that node. This behavior is possible since links in F are Byzantine faulty by assumption.

Consider any node $i \in L$. Recall that E_i^- is the set of all the incoming links at node i. Let E_i' be the subset of links in E_i^- from the nodes in $C \cup R$, i.e.,

$$E_i' = \{(j,i) \mid j \in C \cup R, (j,i) \in \mathcal{E}\}.$$

Since $|F| \leq f$, $|E_i^- \cap F| \leq f$. Moreover, by assumption $C \cup R \not\Rightarrow L$; thus, $|E_i'| \leq f$, and we have $|E_i' - F| \leq |E_i'| \leq f$. Node i will then receive m^- via the links in $E_i^- \cap F$ (if non-empty) and values in $[m, M]$ via the links in $E_i' - F$, and m via the rest of the links, i.e., links in $E_i^- - E_i' - F$.

Consider the following two cases:

- Both $E_i^- \cap F$ and $E_i' - F$ are non-empty:

  In this case, recall that $|E_i^- \cap F| \leq f$ and $|E_i' - F| \leq f$. From node i's perspective, consider two possible scenarios: (a) links in $E_i^- \cap F$ are faulty, and the other links are fault-free, and (b) links in $E_i' - F$ are faulty, and the other links are fault-free.

  In scenario (a), from node i's perspective, all the nodes may have sent values in interval $[m, M]$, but the faulty links have tampered the message so that m^- is delivered to node i. According to the validity property, $v_i[1] \geq m$. On the other hand, in scenario (b), all the nodes may have sent values m^- or m, where $m^- < m$; so $v_i[1] \leq m$, according to the validity property. Since node i does not know whether the correct scenario is (a) or (b), it must update its state to satisfy the validity property in both cases. Thus, it follows that $v_i[1] = m$.

- At most one of $E_i^- \cap F$ and $E_i' - F$ is non-empty:

  Recall that by assumption, $|E_i^- \cap F| \leq f$ and $|E_i' - F| \leq f$. Since at most one of the set is non-empty, $|(E_i^- \cap F) \cup (E_i' - F)| \leq f$. From node i's perspective, it is possible that the links in $(E_i^- \cap F) \cup (E_i' - F)$ are all faulty, and the rest of the links are fault-free. In this situation, all the nodes have sent m to node i, and therefore, $v_i[1]$ must be set to m as per the validity property.

Thus, $v_i[1] = m$ for each node $i \in L$. Similarly, we can show that $v_j[1] = M$ for each node $j \in R$.

Now consider the nodes in set C, if C is non-empty. All the values received by the nodes in C are in $[m, M]$; therefore, their new state must also remain in $[m, M]$, as per the *validity* property.

The above discussion implies that, at the end of iteration 1, the following conditions hold true: (i) state of each node in L is m, (ii) state of each node in R is M, and (iii) state of each node in C is in the interval $[m, M]$. These conditions are identical to the initial conditions listed previously. Then, by a repeated application of the above argument (proof by induction), it follows that for any $t \geq 0$, (i) $v_i[t] = m$ for all $i \in L$; (ii) $v_j[t] = M$ for all $j \in R$; and (iii) $v_k[t] \in [m, M]$ for all $k \in C$.

Since both L and R are non-empty, the ϵ*-agreement* property is not satisfied. A contradiction. □

Theorem 14 shows that *Condition P* is necessary. However, *Condition P* is not intuitive. Below, we state an equivalent condition *Condition S* that is easier to interpret. To facilitate the statement, we introduce the notion of "link-reduced graph". It is different from "reduced graphs" introduced in previous Chapters in the sense that link-reduced graph is constructed by removing links rather than removing nodes.

Definition 20 (Link-Reduced Graph) *For a given graph $G = (\mathcal{V}, \mathcal{E})$ and $F \subset \mathcal{E}$, a graph $G_F = (\mathcal{V}, \mathcal{E}_F)$ is said to be a* link-reduced graph*, if $\mathcal{E}_F$ is obtained by first removing from $\mathcal{E}$ all the links in F, and* then *at each node, removing up to f other incoming links in $\mathcal{E} - F$.*

Note that for a given $G = (\mathcal{V}, \mathcal{E})$ and a given F, multiple link-reduced graphs G_F may exist. Now, we state *Condition S* based on the concept of link-reduce graphs and the notion of source component introduced in Chapter 3.6.4:

Condition S: Consider graph $G = (\mathcal{V}, \mathcal{E})$. For any $F \subseteq \mathcal{E}$ such that $|F| \leq f$, every link-reduced graph G_F obtained as per Definition 20 must contain exactly one *source component*.

Now, we present a key lemma below.

Lemma 39 Condition P *is equivalent to* Condition S.

Proof: We first prove that *Condition P* implies *Condition S.*

By assumption, G contains at least two node, and so does G_F; therefore, at least one source component must exist in G_F. We now prove that G_F cannot contain more than one source component. The proof is by contradiction. Suppose that there exists a subset $F \subset \mathcal{E}$ with $|F| \leq f$, and the link-reduced graph $G_F(\mathcal{V}, \mathcal{E}_F)$ corresponding to F such that the decomposition of G_F includes at least two source components.

Let the sets of nodes in two such source components of G_F be denoted L and R, respectively. Let $C = \mathcal{V} - L - R$. Observe that L, C, R form a partition of the nodes in $\mathcal{V}$. Since L is a source component in G_F, it follows that there are no directed links in $\mathcal{E}_F$ from any node in $C \cup R$ to the nodes in L. Similarly, since R is a source component in G_F, it follows that there are no directed links in $\mathcal{E}_F$ from any node in $L \cup C$ to the nodes in R. These observations, together with the manner in which $\mathcal{E}_F$ is defined, imply that (i) there are at most f links in $\mathcal{E} - F$ from the nodes in $C \cup R$ to each node in L, and (ii) there are at most f links in $\mathcal{E} - F$ from the nodes in $L \cup C$ to each node in R. Therefore, in graph $G' = (\mathcal{V}, \mathcal{E} - F)$, $C \cup R \not\Rightarrow L$ and $L \cup C \not\Rightarrow R$. Thus, $G = (\mathcal{V}, \mathcal{E})$ does not satisfies *Condition P*, since $F \subseteq \mathcal{E}$ and $|F| \leq f$, a contradiction.

Now, we prove that *Condition S* implies *Condition P.*

The proof is by contradiction. Suppose that *Condition P* does not hold for graph $G = (\mathcal{V}, \mathcal{E})$. Thus, there exist a subset $F \subset \mathcal{E}$, where $|F| \leq f$, and a node partition L, C, R, where L and R are both non-empty, such that $C \cup R \not\Rightarrow L$ and $L \cup C \not\Rightarrow R$ in $G' = (\mathcal{V}, \mathcal{E} - F)$.

We now constructed a link-reduced graph $G_F(\mathcal{V}, \mathcal{E}_F)$ corresponding to set F. First, remove all links in F from $\mathcal{E}$. Then since $C \cup R \not\Rightarrow L$, the number of links at each node in L from nodes in $C \cup R$ is at most f; remove all these links. Similarly, for every node $j \in R$, remove all links from nodes in $L \cup C$ to j (recall that by assumption, there are at most f such links). The remaining links form the set $\mathcal{E}_F$. It should be obvious that $G_F(\mathcal{V}, \mathcal{E}_F)$ satisfies Definition 8; hence, G_F is a valid link-reduced graph.

Now, observe that by construction, in the link-reduced graph $G_F(\mathcal{V}, \mathcal{E}_F)$, there are no incoming links to nodes in R from nodes in $L \cup C$; similarly, in G_F, there are no incoming links to nodes in L from nodes in $C \cup R$. It follows that for each $i \in L$, there is no path using links in $\mathcal{E}_F$ from i to nodes in R; similarly, for each $j \in R$, there is no path using links in $\mathcal{E}_F$ from j to nodes in L. Thus, G_F must contain at least two source components. Therefore, the existence of G_F implies that G violates *Condition S*, a contradiction. □

An alternate interpretation of *Condition S* is that in every link-reduced graph G_F, non-fault-tolerant iterative consensus must be possible. We will use this intuition to prove that *Condition S* is sufficient in Chapter 6.5. Then, by Lemma 39, *Condition P* is also sufficient.

Useful Properties Suppose $G = (\mathcal{V}, \mathcal{E})$ satisfies *Condition P* and *Condition S*. We provide two lemmas below to state some properties of $G = (\mathcal{V}, \mathcal{E})$ that are useful for analyzing the iterative algorithm presented later.

Lemma 40 *Suppose that graph $G = (\mathcal{V}, \mathcal{E})$ satisfies* Condition S*. Then, in any link-reduced graph $G_F = (\mathcal{V}, \mathcal{E}_F)$, there exists a node that has a directed path to all the other nodes in $\mathcal{V}$.*

Proof: Recall that *Condition S* states that any link-reduced graph $G_F(\mathcal{V}, \mathcal{E}_F)$ has a single source component. By the definition of source component, any node in the source component (say node s) has directed paths using edges in $\mathcal{E}_F$ to all the other nodes in the source component, since the source component is a strongly connected component. Also, by the uniqueness of the source component, all other strongly connected components in G_F (if any exist) are not source components, and hence reachable from the source component using the edges in $\mathcal{E}_F$. Therefore, node s also has directed paths to all the nodes in $\mathcal{V}$ that are not in the source component as well. Therefore, node s has directed paths to all the other nodes in $\mathcal{V}$. This proves the lemma. □

Lemma 41 *For $f > 0$, if graph $G = (\mathcal{V}, \mathcal{E})$ satisfies* Condition P*, then each node in $\mathcal{V}$ has in-degree at least $2f + 1$, i.e., for each $i \in \mathcal{V}$, $|N_i^-| \geq 2f + 1$.*

Proof: The proof is by contradiction. By assumption in the lemma, $f > 0$, and graph $G = (\mathcal{V}, \mathcal{E})$ satisfies *Condition P*.

Suppose that there exists a node $i \in \mathcal{V}$ such that $|N_i^-| \leq 2f$. Define $L = \{i\}, C = \emptyset$, and $R = \mathcal{V} - \{i\}$. Note that sets L, C, R form a partition of $\mathcal{V}$. Now, define an edge set F such that $F \subseteq \mathcal{E}, |F| \leq f$, and F contains $\min(f, |N_i^-|)$ incoming links from nodes in R to node i.

Observe that $f > 0$, and $|L \cup C| = 1$. Thus, there can be at most 1 link from $L \cup C$ to any node in R in $G' = (\mathcal{V}, \mathcal{E} - F)$. Therefore, $L \cup C \not\Rightarrow R$ in $G' = (\mathcal{V}, \mathcal{E} - F)$. Then, recall that E_i^- is the set of all the node i's incoming links. Since $L = \{i\}$ and $C = \emptyset$, $E_i^- = \{(j, i) \mid j \in R\}$. Also, since $|E_i^-| = |N_i^-| \leq 2f$, and F contains $\min(f, |N_i^-|)$ links in E_i^-, $|E_i^- - F| \leq 2f - f = f$. Therefore, $C \cup R \not\Rightarrow L$ in $G(\mathcal{V}, \mathcal{E} - F)$. Thus, $G = (\mathcal{V}, \mathcal{E})$ does not satisfy *Condition P*, a contradiction. □

6.4 Algorithm 3

We will prove that there exists a correct IABC algorithm – particularly Algorithm 3 below – that satisfies the termination, validity and ϵ-agreement properties provided that the graph $G = (\mathcal{V}, \mathcal{E})$ satisfies *Condition S*. This implies that *Condition P* and *Condition S* ares also sufficient. Algorithm 3 has the iterative structure described in Chapter 4.2, and it is similar to algorithms that were analyzed in previous Chapters, namely Algorithm 1 and 2 (although correctness of the algorithm under the necessary condition – *Conditions P* and *S* – has not been proved previously).

Algorithm 3

1. *Transmit step:* Transmit current state $v_i[t-1]$ on all outgoing edges.

2. *Receive step:* Receive values on all incoming edges. These values form vector $r_i[t]$ of size $|N_i^-|$. If a faulty incoming edge drops the message, then the message value is assumed to be equal to the state at node i, i.e., $v_i[t-1]$.

3. *Update step:* Sort the values in $r_i[t]$ in an increasing order (breaking ties arbitrarily), and eliminate the smallest and largest f values. Let $N_i^*[t]$ denote the set of nodes from whom the remaining $|N_i^-| - 2f$ values in $r_i[t]$ were received. Note that as proved in Lemma 41, each node has at least $2f+1$ incoming neighbors if $f > 0$. Thus, when $f > 0$, $|N_i^*[t]| \geq 1$. Let w_j denote the value received from node $j \in N_i^*[t]$, and for convenience, define $w_i = v_i[t-1]$. Observe that if the link from $j \in N_i^*[t]$ is fault-free, then $w_j = v_j[t-1]$.

   Define

$$v_i[t] \ = \ T_i(r_i[t], v_i[t-1]) \ = \sum_{j \in \{i\} \cup N_i^*[t]} a_i \, w_j \tag{6.1}$$

   where

$$a_i = \frac{1}{|N_i^*[t]| + 1} = \frac{1}{|N_i^-| + 1 - 2f}$$

   The "weight" of each term on the right-hand side of (6.1) is a_i. Note that $|N_i^*[t]| = |N_i^-| - 2f$, and $i \notin N_i^*[t]$ because $(i,i) \notin \mathcal{E}$. Thus, the weights on the right-hand side add to 1. Also, $0 < a_i \leq 1$.

Termination: Each node terminates after completing iteration t_{end}, where t_{end} is a constant defined later in Equation (6.15). The value of t_{end} depends on graph $G = (\mathcal{V}, \mathcal{E})$, constants U and μ defined earlier in Chapter 4.2 and parameter ϵ in ϵ-agreement property.

6.5 Sufficiency (Correctness of Algorithm 3)

We will prove that given a graph $G = (\mathcal{V}, \mathcal{E})$ satisfying *Condition S*, Algorithm 3 is correct, i.e., Algorithm 3 satisfies *termination, validity, ϵ-agreement* properties. Therefore, *Condition S* and *Condition P* are proved to be sufficient. We borrow the matrix analysis from the work on non-fault-tolerant consensus [38, 9, 34]. The proof below follows the same structure in our prior work on node failures in Chapter 5 and [87, 86]; however, such analysis has not been applied in the case of link failures.

In the rest of the discussion, we assume that $G = (\mathcal{V}, \mathcal{F})$ satisfies *Condition S* and *Condition P*. We use transition matrix to represent the *Update* step in Algorithm 3, and show how to use the matrix tools discussed in Chapter 5.5.1 to prove the correctness of Algorithm 3 in $G = (\mathcal{V}, \mathcal{F})$.

In the discussion below, we use boldface upper case letters to denote matrices, rows of matrices, and their elements. For instance, $\mathbf{A}$ denotes a matrix, $\mathbf{A}_i$ denotes the i-th row of matrix $\mathbf{A}$, and $\mathbf{A}_{ij}$ denotes the element at the intersection of the i-th row and the j-th column of matrix $\mathbf{A}$.

Denote by $v[0]$ the column vector consisting of the initial states at all nodes. The i-th element of $v[0]$, $v_i[0]$, is the initial state of node i. Denote by $v[t]$, for $t \geq 1$, the column vector consisting of the states of all nodes at the end of the t-th iteration. The i-th element of vector $v[t]$ is state $v_i[t]$.

For $t \geq 1$, define $F[t]$ to be the set of all faulty links in iteration t. Recall that link (j, i) is said to be faulty in iteration t if the value received by node i is different from what node j sent in iteration t. Then, define N_i^F as the set of all nodes whose outgoing links to node i are faulty in iteration t, i.e.,

$$N_i^F = \{j \mid j \in N_i^-,\ (j, i) \in F[t]\}.^2$$

Now we state the key lemma. In particular, Lemma 42 allows us to use results for non-homogeneous Markov chains to prove the correctness of Algorithm 3. The proof is similar to the proof of Lemma 34 discussed in Chapter 5.5.3.

Lemma 42 *The* Update *step in iteration t $(t \geq 1)$ of Algorithm 3 at the nodes can be expressed as*

$$v[t] = \boldsymbol{M}[t]v[t-1] \tag{6.2}$$

where $\mathbf{M}[t]$ is an $n \times n$ row stochastic transition matrix with the following property: there exist N_i^r, a subset of incoming neighbors at node i of size at most f,[3] and a constant β $(0 < \beta \leq 1)$ that depends only on graph

[2] N_i^F may be different for each iteration t. For simplicity, the notation does not explicitly represent this dependence.

[3] Intuitively, N_i^r corresponds to the links removed in some link-reduced graph. Thus, the superscript r in the notation stands

$G = (\mathcal{V}, \mathcal{E})$ such that for each $i \in \mathcal{V}$, and for all $j \in \{i\} \cup (N_i^- - N_i^F - N_i^r)$,

$$\boldsymbol{M}_{ij}[t] \geq \beta.$$

Proof: We prove the correctness of Lemma 42 by constructing $\mathbf{M}_i[t]$ for $1 \leq i \leq n$ that satisfies the conditions in Lemma 42. Recall that $F[t]$ denotes the set of faulty links in the t-th iteration.

Consider a node i in iteration t ($t \geq 1$). In the *Update* step of Algorithm 1, recall that the smallest and the largest f values are removed from $r_i[t]$ by node i. Denote by $\mathcal{S}$ and $\mathcal{L}$, respectively, the set of nodes[4] from whom the smallest and the largest f values were received by node i in iteration t. Define sets $\mathcal{S}_g$ and $\mathcal{L}_g$ to be subsets of $\mathcal{S}$ and $\mathcal{L}$ that contain all the nodes from whom node i receives the correct value in $\mathcal{S}$ and $\mathcal{L}$, respectively. That is, $\mathcal{S}_g = \{j \mid j \in \mathcal{S},\ (j,i) \in \mathcal{E} - F[t]\}$ and $\mathcal{L}_g = \{j \mid j \in \mathcal{L},\ (j,i) \in \mathcal{E} - F[t]\}$.

Construction of $\mathbf{M}_i[t]$ differs somewhat depending on whether sets $\mathcal{S}_g, \mathcal{L}_g$ and N_i^F are empty or not. We divide the possibilities into 3 separate cases:

- Case I: $\mathcal{S}_g \neq \emptyset, \mathcal{L}_g \neq \emptyset$, and $N_i^F \neq \emptyset$.
- Case II: $\mathcal{S}_g \neq \emptyset, \mathcal{L}_g \neq \emptyset$, and $N_i^F = \emptyset$.
- Case III: at most one of $\mathcal{S}_g$ and $\mathcal{L}_g$, and $N_i^F = \emptyset$.

Observe that if $\mathcal{S}_g$ ($\mathcal{L}_g$) is empty, then $N_i^F = \emptyset$ and $\mathcal{L} = \mathcal{L}_g$ ($\mathcal{S} = \mathcal{S}_g$), since there are at most f faulty links and $|\mathcal{S}| = |\mathcal{L}| = f$. Therefore, the 3 cases above cover all the possible scenarios.

Case I

In Case I, $\mathcal{S}_g \neq \emptyset, \mathcal{L}_g \neq \emptyset$, and $N_i^F \neq \emptyset$. Let $m_{\mathcal{S}}$ and $m_{\mathcal{L}}$ be defined as shown below. Recall that the incoming links from the nodes in $\mathcal{S}_g$ and $\mathcal{L}_g$ to node i are all fault-free, and therefore, for any node $j \in \mathcal{S}_g \cup \mathcal{L}_g$, $w_j = v_j[t-1]$ (in the notation of Algorithm 1). That is, the value received by node i from node j is exactly the state at node j in iteration $t-1$.

$$m_{\mathcal{S}} = \frac{\sum_{j \in \mathcal{S}_g} v_j[t-1]}{|\mathcal{S}_g|} \qquad \text{and} \qquad m_{\mathcal{L}} = \frac{\sum_{j \in \mathcal{L}_g} v_j[t-1]}{|\mathcal{L}_g|}$$

Now, consider any node $k \in N_i^F$. By the definition of sets $\mathcal{S}_g$ and $\mathcal{L}_g$, $m_{\mathcal{S}} \leq w_k \leq m_{\mathcal{L}}$. Therefore, we can find weights $S_k \geq 0$ and $L_k \geq 0$ such that $S_k + L_k = 1$, and

for "removed." Also, N_i^r may be different for each t. For simplicity, the notation does not explicitly represent this dependence.

[4] Although $\mathcal{S}$ and $\mathcal{L}$ may be different for each iteration t, for simplicity, we do not explicitly represent this dependence on t in the notations $\mathcal{S}$ and $\mathcal{L}$.

$$\begin{aligned} w_k &= S_k \; m_{\mathcal{S}} + L_k \; m_{\mathcal{L}} && (6.3) \\ &= \frac{S_k}{|\mathcal{S}_g|} \sum_{j \in \mathcal{S}_g} v_j[t-1] + \frac{L_k}{|\mathcal{L}_g|} \sum_{j \in \mathcal{L}_g} v_j[t-1] && (6.4) \end{aligned}$$

Clearly, at least one of S_k and L_k must be $\geq 1/2$.

We now define elements $\mathbf{M}_{ij}[t]$ of row $\mathbf{M}_i[t]$:

- For $j \in N_i^*[t] - N_i^F$: In this case, either the edge (j,i) is fault-free, or $j = i$. For each such j, define $\mathbf{M}_{ij}[t] = a_i$. This is obtained by observing in (6.1) that the contribution of such a node j to the new state $v_i[t]$ is $a_i \; w_j = a_i \; v_j[t-1]$.

  The elements of $\mathbf{M}_i[t]$ defined here add up to

  $$|N_i^*[t] - N_i^F| \; a_i$$

- For $j \in \mathcal{S}_g \cup \mathcal{L}_g$: In this case, the edge (j,i) is a fault-free.

  For each $j \in \mathcal{S}_g$,

  $$\mathbf{M}_{ij}[t] \;=\; a_i \sum_{k \in N_i^F} \frac{S_k}{|\mathcal{S}_g|}$$

  and for each node $j \in \mathcal{L}_g$,

  $$\mathbf{M}_{ij}[t] \;=\; a_i \sum_{k \in N_i^F} \frac{L_k}{|\mathcal{L}_g|}$$

  To obtain these two expressions, we represent value w_k sent via faulty link (k,i) for each $k \in N_i^F$ using (6.4). Recall that this node k contributes $a_i w_k$ to (6.1). The above two expressions are then obtained by summing (6.4) over all the nodes in N_i^F, and replacing this sum by equivalent contributions by nodes in $\mathcal{S}_g$ and $\mathcal{L}_g$.

  The elements of $\mathbf{M}_i[t]$ defined here add up to $a_i \sum_{k \in N_i^F} (S_k + L_k) = |N_i^F| \; a_i$

- For $j \in \mathcal{V} - ((N_i^* - N_i^F) \cup \mathcal{S}_g \cup \mathcal{L}_g)$: These nodes have not yet been considered above. For each such node j, define $\mathbf{M}_{ij}[t] = 0$.

With the above definition of $\mathbf{M}_i[t]$, it should be easy to see that $\mathbf{M}_i[t] \, v[t-1]$ is, in fact, identical to $v_i[t]$ obtained using (6.1). Thus, the above construction of $\mathbf{M}_i[t]$ results in the values sent via faulty links to (6.1) being replaced by an equivalent contribution from the nodes in $\mathcal{L}_g$ and $\mathcal{S}_g$.

Properties of $\boldsymbol{M}_i[t]$:

First, we show that $\mathbf{M}[t]$ is row stochastic. Observe that all the elements of $\mathbf{M}_i[t]$ are non-negative. Also, all the elements of $\mathbf{M}_i[t]$ above add up to

$$|N_i^*[t] - N_i^F|\ a_i + |N_i^F|\ a_i = |N_i^*[t]|\ a_i = 1$$

because $a_i = 1/|N_i^*[t]|$ as defined in Algorithm 1. Thus, $\mathbf{M}_i[t]$ is a stochastic row vector.

Recall that from the above discussion, for $k \in N_i^F$, one of S_k and L_k must be $\geq 1/2$. Without loss of generality, assume that $S_s \geq 1/2$ for all nodes $s \in N_i^F$. Consequently, for each node $j \in \mathcal{S}_g$, $\mathbf{M}_{ij}[t] \geq \frac{a_i}{|\mathcal{S}_g|} S_s \geq \frac{a_i}{2|\mathcal{S}_g|}$. Also, for each node j in $N_i^*[t] - N_i^F$, $\mathbf{M}_{ij}[t] = a_i$. Thus, if β is chosen such that

$$0 < \beta \leq \frac{a_i}{2|\mathcal{S}_g|} \tag{6.5}$$

and N_i^r is defined to be $\mathcal{L}_g$, then the condition in the lemma holds for node i. That is, for all $j \in \{i\} \cup (N_i^- - N_i^F - N_i^r)$,

$$\mathbf{M}_{ij}[t] \geq \beta$$

Case II

Now, we consider the case when $\mathcal{S}_g \neq \emptyset, \mathcal{L}_g \neq \emptyset$, and $N_i^F = \emptyset$. That is, when each of $\mathcal{S}$ and $\mathcal{L}$ contains at least one node from which the node i receives correct value, and node i receives correct value(s) from all the node(s) in $N_i^*[t]$. In fact, the analysis of Case II is very similar to the analysis presented above in Case I. We now discuss how the analysis of Case I can be applied to Case II. Rewrite (6.1) as follows:

$$\begin{aligned} v_i[t] &= \frac{a_i}{2} v_i[t-1] + \frac{a_i}{2} v_i[t-1] + \sum_{j \in N_i^*[t] - \{i\}} a_i w_j && (6.6) \\ &= a_i w_z + a_i w_i + \sum_{j \in N_i^*[t] - \{i\}} a_i w_j && (6.7) \end{aligned}$$

In the above equation, z is to be viewed as a "virtual" incoming neighbor of node i, which has sent value $w_z = \frac{v_i[t-1]}{2}$ to node i in iteration t. With the above rewriting of state update, the value received by node

i from itself should be viewed as $w_i = \frac{v_i[t-1]}{2}$ instead of $v_i[t-1]$. With this transformation, Case II now becomes identical to Case I, with virtual node z being treated as an incoming neighbor of node i.

In essence, a part of node i's contribution (half, to be precise) is now replaced by equivalent contribution by nodes in $\mathcal{L}_g$ and $\mathcal{S}_g$. We now define elements $\mathbf{M}_{ij}[t]$ of row $\mathbf{M}_i[t]$:

- For $j = i$: $\mathbf{M}_{ij}[t] = \frac{a_i}{2}$. This is obtained by observing in (6.1) that node i's contribution to the new state $v_i[t]$ is $a_i \frac{v_i[t-1]}{2}$.

- For $j \in N_i^*[t] - \{i\}$: In this case, j is a node from which node i receives correct value. For each such j, define $\mathbf{M}_{ij}[t] = a_i$. This is obtained by observing in (6.1) that the contribution of node j to the new state $v_i[t]$ is $a_i w_j = a_i v_j[t-1]$.

- For $j \in \mathcal{S}_g \cup \mathcal{L}_g$: In this case, j is a node in $\mathcal{S}$ or $\mathcal{L}$ from which node i receives correct value.

  For each $j \in \mathcal{S}_g$,
$$\mathbf{M}_{ij}[t] \;=\; \frac{a_i}{2}\,\frac{S_z}{|\mathcal{S}_g|}$$
  and for each node $j \in \mathcal{L}_g$,
$$\mathbf{M}_{ij}[t] \;=\; \frac{a_i}{2}\,\frac{L_z}{|\mathcal{L}_g|}$$
  where S_z and L_z are chosen such that $S_z + L_z = 1$ and $w_z = \frac{v_i[t-1]}{2} = \frac{S_z}{2} m_{\mathcal{S}} + \frac{L_z}{2} m_{\mathcal{L}}$. Note that such S_z and L_z exist because by definition of $\mathcal{S}_g$ and $\mathcal{L}_g$, $v_i[t-1] \geq w_j$, $\forall j \in S_g$ and $v_i[t-1] \leq w_j$, $\forall j \in L_g$. Then the two expressions above are obtained by replacing the contribution of the virtual node z by an equivalent contribution by the nodes in $\mathcal{S}_g$ and $\mathcal{L}_g$, respectively.

- For $j \in \mathcal{V} - (N_i^*[t] \cup \mathcal{S}_g \cup \mathcal{L}_g)$: These nodes have not yet been considered above. For each such node j, define $\mathbf{M}_{ij}[t] = 0$.

Properties of $\boldsymbol{M}_i[t]$:

By argument similar to that in *Case I*, $\mathbf{M}_i[t]$ is row stochastic. Without loss of generality, suppose that $S_z \geq 1/2$. Then for each node $j \in \mathcal{S}_g$, $\mathbf{M}_{ij}[t] = \frac{a_i}{2|\mathcal{S}_g|} S_z \geq \frac{a_i}{4|\mathcal{S}_g|}$. Also, for node j in $N_i^*[t] - \{i\}$, $\mathbf{M}_{ij}[t] = a_i$, and $\mathbf{M}_{ii}[t] = \frac{a_i}{2}$. Recall that by definition, $|\mathcal{S}_g| \geq 1$. Hence, if β is chosen such that

$$0 < \beta \leq \frac{a_i}{4|\mathcal{S}_g|} \tag{6.8}$$

and N_i^r is defined to be equal to $\mathcal{L}_g$, then the condition in the Lemma 42 holds for node i. That is, $\mathbf{M}_{ij}[t] \geq \beta$ for $j \in \{i\} \cup (N_i^- - N_i^F - N_i^r)$.

Case III

Here, we consider the case when at most one of $\mathcal{S}_g$ and $\mathcal{L}_g$ is empty, and $N_i^F = \emptyset$. Without loss of generality, suppose that $\mathcal{S}$ contains only nodes whose outgoing links to node i is faulty in iteration t, i.e., $\mathcal{S} = \{j \mid (j,i) \in F[t]\}$. Since there are at most f faulty links and $|\mathcal{S}| = f$, $\mathcal{L} = \mathcal{L}_g$. That is, the value received from each node in $\mathcal{L}$ by node i is correct.

In this case, define $\mathbf{M}_{ij}[t] = a_i$ for $j \in N_i^*[t]$; define $\mathbf{M}_{ij} = 0$ for all other nodes j.

Properties of $\boldsymbol{M}_i[t]$:

All the elements of $\mathbf{M}_i[t]$ are non-negative. The elements of $\mathbf{M}_i[t]$ defined above add up to

$$|N_i^*[t]|\ a_i = 1$$

Thus, $\mathbf{M}_i[t]$ is a stochastic row vector.

In Case III, recall that for any node j in $N_i^*[t]$, $\mathbf{M}_{ij}[t] = a_i$. Thus, if β is chosen such that

$$0 < \beta \le a_i \tag{6.9}$$

and N_i^r is defined to be equal to $\mathcal{L}$, then the condition in the Lemma 42 holds for node i.

Putting All Cases Together

Now, let us consider Cases I-III together. From the definition of a_i in Algorithm 1, observe that $a_i \geq \frac{1}{|N_i^-|+1}$ (because $f \geq 0$). Let us define

$$\alpha = \min_{i \in \mathcal{V}} \frac{1}{|N_i^-|+1}$$

Moreover, observe that $|\mathcal{S}_g| \le n$ and $|\mathcal{L}_g| \le n$. Then define β as

$$\beta = \frac{\alpha}{4n} \tag{6.10}$$

This definition satisfies constraints on β in Cases I through III (conditions (6.5), (6.8) and (6.9)). Thus, Lemma 42 holds for all three cases with this choice in (6.10). □

Matrix $\mathbf{M}[t]$ is said to be a transition matrix for iteration t. As the lemma states above, $\mathbf{M}[t]$ is a row stochastic matrix. The proof of Lemma 42 shows how to construct a suitable row stochastic matrix $\mathbf{M}[t]$ for each iteration t. $\mathbf{M}[t]$ depends not only on t but also on the behavior of the faulty links in iteration t.

Theorem 15 *Algorithm 3 satisfies the* Termination, Validity*, and* ϵ-agreement *properties.*

Proof: Chapters 6.5.1, 6.5.2 and 6.5.3 provide the proof that Algorithm 3 satisfies the three properties, respectively, in the presence of Byzantine links. This proof follows a structure used to prove correctness of other consensus algorithms in our prior work presented in Chapter 5 and [87, 86]. However, there is a difference between proving ϵ-agreement property under link failures, and proving convergence property under node failures. We include the proof here for completeness. □

6.5.1 Validity Property

Observe that $\mathbf{M}[t+1](\mathbf{M}[t]v[t-1]) = (\mathbf{M}[t+1]\mathbf{M}[t])v[t-1]$. Therefore, by repeated application of (6.2), we obtain for $t \geq 1$,

$$v[t] = (\Pi_{u=1}^{t}\mathbf{M}[u])v[0] \tag{6.11}$$

Since each $\mathbf{M}[u]$ is row stochastic as shown in Lemma 42, the matrix product $\Pi_{u=1}^{t}\mathbf{M}[u]$ is also a row stochastic matrix. Thus, (6.11) implies that the state of each node i at the end of iteration t can be expressed as a convex combination of the initial states at all the nodes. Therefore, the validity property is satisfied.

6.5.2 Termination Property

Algorithm 3 terminates after t_{end} iterations, where t_{end} is a finite constant depending only on $G = (\mathcal{V}, \mathcal{E}), U, \mu$, and ϵ. Recall that U and μ are defined as upper and lower bounds of the initial inputs at all nodes, respectively. Therefore, trivially, the algorithm satisfies the termination property. Later, using (6.15), we define a suitable value for t_{end}.

6.5.3 ϵ-agreement Property

Denote by R_F the set of all the link-reduced graph of $G = (\mathcal{V}, \mathcal{E})$ corresponding to some faulty link set F. Let

$$r = \sum_{F \subset \mathcal{E},\ |F| \leq f} |R_F|$$

Note that r only depends on $G = (\mathcal{V}, \mathcal{E})$ and f, and is a finite integer.

Consider iteration t ($t \geq 1$). Recall that $F[t]$ denotes the set of faulty links in iteration t. Then for each link-reduced graph $H[t] \in R_{F[t]}$, define connectivity matrix $\mathbf{H}[t]$ as follows, where $1 \leq i, j \leq n$:

- $\mathbf{H}_{ij}[t] = 1$, if either $j = i$, or edge (j, i) exists in link-reduced graph H;
- $\mathbf{H}_{ij}[t] = 0$, otherwise.

Thus, the non-zero elements of row $\mathbf{H}_i[t]$ correspond to the incoming links at node i in the link-reduced graph $H[t]$, or the self-loop at i. Observe that $\mathbf{H}[t]$ has a non-zero diagonal.

Based on *Condition S* and Lemmas 40, 42, we can show the following key lemmas.

Lemma 43 *For any $H[t] \in R_{F[t]}$, and $k \geq n$, $\mathbf{H}^k[t]$ has at least one non-zero column, i.e., a column with all elements non-zero.*

Proof: $G(\mathcal{V}, \mathcal{E})$ satisfies the *Condition S*. Therefore, by Lemma 40, there exists at least one node p in the link-reduced graph $H[t]$ that has directed paths to all the nodes in $H[t]$ (consisting of the edges in $H[t]$). $\mathbf{H}^k_{jp}[t]$ of product $\mathbf{H}^k[t]$ is 1 if and only if node p has a directed path to node j consisting of at most k edges in $H[t]$. Since the length of the path from p to any other node in $H[t]$ is at most n, and p has directed paths to all the nodes, for $k \geq n$ the p-th column of matrix $\mathbf{H}^k[t]$ will be non-zero.[5] □

Then, Lemma 43 can be used to prove the following lemma.

Lemma 44 *For any $z \geq 1$, at least one column in the matrix product $\Pi_{t=u}^{u+rn-1} \mathbf{H}[t]$ is non-zero.*

Proof: Since $\Pi_{t=u}^{u+rn-1} \mathbf{H}[t]$ consists of rn connectivity matrices corresponding to link-reduced graphs, and the number of all link-reduced graphs for F ($|F| \leq f$) is r, connectivity matrices corresponding to at least one link-reduced graph, say matrix $\mathbf{H}_*$, will appear in the above product at least n times.

Now observe that: (i) By Lemma 43, $\mathbf{H}^n_*$ contains a non-zero column, say the k-th column is non-zero, and (ii) by definition, all the $\mathbf{H}[t]$ matrices in the product contain a non-zero diagonal. These two observations together imply that the k-th column in the above product is non-zero.[6] □

For matrices $\mathbf{A}$ and $\mathbf{B}$ of identical dimension, we say that $\mathbf{A} \leq \mathbf{B}$ iff $\mathbf{A}_{ij} \leq \mathbf{B}_{ij}$ for all i, j. Lemma below relates the transition matrices with the connectivity matrices. Constant β used in the lemma below was introduced in Lemma 42.

[5] That is, all the elements of the column will be non-zero. Also, such a non-zero column will exist in $\mathbf{H}^{n-1}[t]$, too. We use the loose bound of n to simplify the presentation.

[6] The product $\Pi_{t=u}^{u+rn-1} \mathbf{H}[t]$ can be viewed as the product of n instances of $\mathbf{H}_*$ "interspersed" with matrices with non-zero diagonals.

Lemma 45 *For any $t \geq 1$, there exists a link-reduced graph $H[t] \in R_{F[t]}$ such that $\beta\mathbf{H[t]} \leq \mathbf{M}[t]$, where $\mathbf{H}[t]$ is the connectivity matrix for $H[t]$.*

Proof: First, let us construct a link-reduced graph $H[t]$ by first removing $F[t]$ from $G(\mathcal{V}, \mathcal{E})$. Recall that $F[t]$ is the set of faulty links in iteration t. Then for each i, remove a set of at most f node i's incoming links as defined in Lemma 42 (N_i^r). As a result, we have obtained a link-reduced graph $H[t]$ such that $\mathbf{M}_{ij}[t] \geq \beta$, if $j = i$ or edge (j, i) is in the link-reduced graph $H[t]$.

Denote by $\mathbf{H}[t]$ the connectivity matrix for the link-reduced graph $H[t]$. Then, $\mathbf{H}_{ij}[t]$ denotes the element in i-th row and j-th column of $\mathbf{H}[t]$. By definition of the connectivity matrix, we know that $\mathbf{H}_{ij}[t] = 1$, if $j = i$ or edge (j, i) is in the link-reduced graph; otherwise, $\mathbf{H}_{ij}[t] = 0$.

The statement in the lemma then follows from the above two observations. □

Let us now define a sequence of matrices $\mathbf{Q}(i)$, $i \geq 1$, such that each of these matrices is a product of rn of the $\mathbf{M}[t]$ matrices. Specifically,

$$\mathbf{Q}(i) \quad = \quad \Pi_{t=(i-1)rn+1}^{irn} \, \mathbf{M}[t] \tag{6.12}$$

From (6.11) and (6.12) observe that

$$v[krn] \quad = \quad \left(\Pi_{i=1}^{k} \, \mathbf{Q}(i)\right) \; v[0] \tag{6.13}$$

Based on (6.13), Lemmas 42, 44, and 45, we can show the following lemma.

Lemma 46 *For $i \geq 1$, $\mathbf{Q}(i)$ is a row stochastic matrix, and*

$$\lambda(\mathbf{Q}(i)) \leq 1 - \beta^{rn}.$$

Proof:

$\mathbf{Q}(i)$ is a product of row stochastic matrices ($\mathbf{M}[t]$); therefore, $\mathbf{Q}(i)$ is row stochastic. From Lemma 45, for each $t \geq 1$,

$$\beta \, \mathbf{H}[t] \; \leq \; \mathbf{M}[t]$$

Therefore,

$$\beta^{rn} \, \Pi_{t=(i-1)rn+1}^{irn} \, \mathbf{H}[t] \; \leq \; \Pi_{t=(i-1)rn+1}^{irn} \, \mathbf{M}[t] \; = \; \mathbf{Q}(i)$$

By using $u = (i-1)n + 1$ in Lemma 44, we conclude that the matrix product on the left side of the above inequality contains a non-zero column. Therefore, since $\beta > 0$, $\mathbf{Q}(i)$ on the right side of the inequality also contains a non-zero column.

Observe that rn is finite, and hence, β^{rn} is non-zero. Since the non-zero terms in $\mathbf{H}[t]$ matrices are all 1, the non-zero elements in $\Pi_{t=(i-1)rn+1}^{irn}\mathbf{H}[t]$ must each be ≥ 1. Therefore, there exists a non-zero column in $\mathbf{Q}(i)$ with all the elements in the column being $\geq \beta^{rn}$. Therefore, by Lemma 33, $\lambda(\mathbf{Q}(i)) \leq 1 - \beta^{rn}$, and $\mathbf{Q}(i)$ is a scrambling matrix. □

Let us now continue with the proof of ϵ-agreement. Consider the coefficient of ergodicity $\delta(\Pi_{u=1}^{t}\mathbf{M}[u])$.

$$\begin{aligned}
\delta(\Pi_{u=1}^{t}\mathbf{M}[u]) &= \delta\left(\left(\Pi_{u=(\lfloor \frac{t}{rn} \rfloor)rn+1}^{t}\mathbf{M}[u]\right)\left(\Pi_{u=1}^{\lfloor \frac{t}{rn} \rfloor}\mathbf{Q}(u)\right)\right) \quad \text{by definition of } \mathbf{Q}(u)\\
&\leq \lambda\left(\Pi_{u=(\lfloor \frac{t}{rn} \rfloor)rn+1}^{t}\mathbf{M}[u]\right)\left(\Pi_{u=1}^{\lfloor \frac{t}{rn} \rfloor}\lambda\left(\mathbf{Q}(u)\right)\right) \quad \text{by Lemma 32}\\
&\leq \Pi_{u=1}^{\lfloor \frac{t}{rn} \rfloor}\lambda\left(\mathbf{Q}(u)\right) \quad \text{because } \lambda(\cdot) \leq 1\\
&\leq (1-\beta^{rn})^{\lfloor \frac{t}{rn} \rfloor} \quad \text{by Lemma 46} \qquad (6.14)
\end{aligned}$$

Observe that the upper bound on right side of (6.14) depends only on graph $G = (\mathcal{V}, \mathcal{E})$ and t, and is independent of the input states, and the behavior of the faulty links. Moreover, the upper bound on the right side of (6.14) is a non-increasing function of t. Define t_{end} as the smallest positive integer such that the right hand side of (6.14) is smaller than $\frac{\epsilon}{n\max(|U|,|\mu|)}$. Recall that U and μ are defined as the upper and lower bound of the inputs at all nodes. Thus,

$$\delta(\Pi_{u=1}^{t_{end}}\mathbf{M}[u]) \leq (1-\beta^{rn})^{\lfloor \frac{t_{end}}{rn} \rfloor} < \frac{\epsilon}{n\max(|U|,|\mu|)} \qquad (6.15)$$

Recall that β and r depend only on $G = (\mathcal{V}, \mathcal{E})$. Thus, t_{end} depends only on graph $G = (\mathcal{V}, \mathcal{E})$, and constants U, μ and ϵ.

By construction, $\Pi_{u=1}^{t}\mathbf{M}[u]$ is an $n \times n$ row stochastic matrix. Let $\mathbf{M}^* = \Pi_{u=1}^{t}\mathbf{M}[u]$. We omit time index $[t]$ from the notation $\mathbf{M}^*$ for simplicity. From (6.11), we have $v_j[t] = \mathbf{M}_j^* v[0]$. That is, the state of any node j can be obtained as the product of the j-th row of $\mathbf{M}^*$ and $v[0]$. Now, consider any two nodes j, k, we have

$$
\begin{aligned}
|v_j[t] - v_k[t]| &= |\mathbf{M}_j^* v[0] - \mathbf{M}_k^* v[0]| \\
&= |\Sigma_{i=1}^n \mathbf{M}_{ji}^* v_i[0] - \Sigma_{i=1}^n \mathbf{M}_{ki}^* v_i[0]| \\
&= |\Sigma_{i=1}^n \left(\mathbf{M}_{ji}^* - \mathbf{M}_{ki}^*\right) v_i[0]| \\
&\leq \Sigma_{i=1}^n |\mathbf{M}_{ji}^* - \mathbf{M}_{ki}^*||v_i[0]| \\
&\leq \Sigma_{i=1}^n \delta(\mathbf{M}^*)|v_i[0]| \\
&\leq n\delta(\mathbf{M}^*)\max(|U|, |\mu|) \\
&\leq n\delta(\Pi_{u=1}^t \mathbf{M}[u])\max(|U|, |\mu|) \qquad (6.16)
\end{aligned}
$$

Therefore, by (6.15) and (6.16), we have

$$
|v_j[t_{end}] - v_k[t_{end}]| < \epsilon \qquad (6.17)
$$

Since the output of the nodes equal its state at termination (after t_{end} iterations). Thus, (6.17) implies that Algorithm 3 satisfies the ϵ-agreement property.

6.6 Summary

This Chapter explores approximate consensus problem under transient Byzantine link failure model [64, 65]. We address a particular class of iterative algorithms in arbitrary directed graphs, and prove the *tight* necessary and sufficient condition for the graphs to be able to solve the approximate consensus problem in the presence of Byzantine links iteratively.

Chapter 7

Iterative Approximate Crash-tolerant Consensus in Asynchronous Systems

7.1 Introduction

Most of the previous Chapters considers fault-tolerant consensus in synchronous systems (with the exception of Chapters 3.5 and 4.6). This Chapter explores iterative approximate consensus in asynchronous systems in which up to f nodes may suffer crash failures (f-total fault model). Recall that while exact consensus is not solvable in asynchronous systems [33], approximate fault-tolerant consensus can be solved using iterative algorithms. Thus, we only consider approximate consensus here. This Chapter differs from previous relevant Chapters as follows:

- Chapter 3.5 considers crash-tolerant consensus in asynchronous systems, using *general* algorithms. In this Chapter, we are interested in a *restricted class* of iterative algorithms that maintain only a small amount of memory across iterations, e.g., the algorithms do not require the nodes to have a knowledge of the entire network topology.
- Chapter 4.6 considers iterative algorithms tolerating Byzantine faults; whereas, in this Chapter, we focus on crash failures.
- Chapter 6 considers link faults; whereas, in this Chapter, all the links are assumed to be fault-free, except that message transmission between any pair of nodes may be delayed indefinitely due to asynchronous message delay in this Chapter, i.e., asynchronous systems.

We have already discussed the properties of the iterative algorithms in Chapter 4.2. We include them below for readers' ease. Note that due to the assumption of crash faults, the validity property is slightly different form the one for Byzantine fault.

- *Initial state* of each node is equal to a real-valued *input* provided to that node.
- *Termination*: The algorithm terminates in finite number of iterations.

- *Validity*: After each iteration of the algorithm, the state of each node must stay in the *convex hull* of the states of all the nodes at the end of the *previous* iteration.

- ϵ-*agreement*: For any $\epsilon > 0$, when the algorithm terminates, the difference between the outputs at any pair of nodes is guaranteed to be within ϵ.

This Chapter considers iterative crash-tolerant consensus in *asynchronous* systems in arbitrary *directed* networks. The algorithms presented bear some similarity to the asynchronous iterative Byzantine consensus algorithm discussed in Chapter 4.6; however, the analysis is different due to the nature of different faulty behavior. Also, as we will see later, to tolerate Byzantine faults, communication graph requires richer network.

For simplicity, we assume that the input at each node is some *real number* in the range $[0, K]$. Note that if $K < \epsilon$, then the problem is trivial, so K is assumed to be $\geq \epsilon$.

IAC Algorithms We are interested in the IAC (Iterative Approximate Consensus) algorithms. Note that IAC's three-step structure is the same as the ones of IABC (described in Chapter 4.2). However, the correctness definition is different due to the nature of crash faults. To accommodate asynchronous systems, we adopt the same approach in [29], which has two main differences from iterative consensus algorithms in synchronous systems (described in Chapters 4, 5, and 6): (i) the messages containing states are now tagged by the round index to which the states correspond, and (ii) each node i waits to receive only $|N_i^-| - f$ messages containing states from round $t-1$ before computing the new state in round t. Due to the asynchrony assumption, different nodes may potentially perform their t-th round at very different real times. Thus, the main difference between iteration and round is as following:

- Iteration is defined as a duration of fixed amount of real-time units. Hence, every node will be in the same iteration at any given real time.

- Round is defined as the time that each node updates its value[1]. Hence, every node may be in totally different rounds at any given real time in asynchronous systems.

Each node i maintains state v_i, with $v_i[t]$ denoting the state of node i at the *end* of the t-th round of the algorithm ($t \geq 0$). Initial state of node i, $v_i[0]$, is equal to the initial *input* provided to node i. At the *start* of the t-th round ($t > 0$), the state of node i is $v_i[t-1]$. We assume that the input at each node is lower bounded by a constant μ and upper bounded by a constant U. The iterative algorithm may terminate after a number of rounds that is a function of μ and U. μ and U are assumed to be known a priori.

[1] With a slight abuse of terminology, we will use "value" and "state" interchangeably in this dissertation.

The IAC (Iterative Approximate Consensus) algorithms of interest will require each node i to perform the following three steps in round t, where $t > 0$.

1. *Transmit step:* Transmit current state, namely $v_i[t-1]$, on all outgoing edges (to nodes in N_i^+). The message is tagged by index $t-1$.
2. *Receive step:* Wait until the first $|N_i^-| - f$ messages tagged by index $t-1$ are received on the incoming edges (breaking ties arbitrarily). Values received in these messages form vector $r_i[t]$ of size $|N_i^-| - f$.
3. *Update step:* Node i updates its state using a transition function Z_i as follows. Z_i is a part of the specification of the algorithm, and takes as input the vector $r_i[t]$ and state $v_i[t-1]$.

$$v_i[t] \quad = \quad Z_i \; (\; r_i[t] \, , \, v_i[t-1] \;) \tag{7.1}$$

Finally, the output is set to the state at termination. Note that the IAC algorithms are non-blocking, since at the Receive step, each fault-free node i can eventually receive $|N_i^-| - f$ messages, since by assumption, up to f nodes may crash.

Denote by $\mathcal{F}[t]$ the set of crashed node in the end of round t. Note that each node $i \in \mathcal{V} - \mathcal{F}[t]$ have corrected computed value $v_i[t]$ by assumption. The following properties must be satisfied by a correct IAC algorithm in the presence of up to f crash failures:

- **Termination**: the algorithm terminates in finite number of rounds.
- **Validity:**[2] $\forall t > 0$, and $\forall i \in \mathcal{V} - \mathcal{F}[t]$,

$$\min_{j \in \mathcal{V} - \mathcal{F}[t-1]} v_j[t-1] \le v_i[t] \le \max_{j \in \mathcal{V} - \mathcal{F}[t-1]} v_j[t-1]$$

- **ϵ-agreement:** If the algorithm terminates after t_{end} rounds, then $\forall i, j \in \mathcal{V} - \mathcal{F}[t_{end}]$,

$$|v_i[t_{end}] - v_j[t_{end}]| < \epsilon.$$

For a given communication graph $G(\mathcal{V}, \mathcal{E})$, the objective in this Chapter is to identify the necessary and sufficient conditions in graph G for the existence of a *correct* IAC algorithm (i.e., an algorithm satisfying the above properties).

[2] For brevity of presentation, assume that $\mathcal{F}[0] = \emptyset$, i.e., no node has crashed before the start of the algorithm; however, this simplification does not affect the results of this Chapter, and the analysis can be extended to a more general case.

7.2 Necessary Condition

For a correct IAC algorithm to exist in asynchronous systems with crash faults, the graph $G(\mathcal{V},\mathcal{E})$ must satisfy the necessary condition proved in this section. Recall that in Chapter 4.3, we have introduced relations $\Rightarrow$ and $\not\Rightarrow$ in Definition 10. This Chapter will use these relations to define the tight condition.

Condition ICCA : Consider graph $G(\mathcal{V},\mathcal{E})$. Let sets L, C, R form a partition of $\mathcal{V}$, such that both L and R are non-empty. Then, at least one of the two conditions below must be true: (i) $C \cup R \Rightarrow L$ or (ii) $L \cup C \Rightarrow R$.

Condition ICCA is the abbreviation of Iterative Crash-tolerant Consensus in Asynchronous systems. The proof below is similar to the necessity proof of Condition CCA presented in Chapter 3.5 (the *tight* condition of the graph for achieving approximate consensus using *general* algorithms in asynchronous systems). The differ is the set of links that we choose to be *slow* links.

Theorem 16 *Suppose that a correct IABC algorithm exists for $G(\mathcal{V},\mathcal{E})$. Then G satisfies* Condition ICCA.

Proof: The proof is by contradiction. Suppose that there exists a correct IAC algorithm in $G(\mathcal{V},\mathcal{E})$, but $G(\mathcal{V},\mathcal{E})$ does not satisfy *Condition ICCA*. That is, there exists a node partition L, C, R such that L and R are non-empty, and $L \cup C \not\Rightarrow R$ and $C \cup R \not\Rightarrow L$. By Definition 10, all the nodes in $L \cup R$ have at most f incoming neighbors. For any pair of nodes i, j such that $i \in L$ and $j \in R$, let $O(i)$ denote the set of nodes in $C \cup R$ that have outgoing links to node i, i.e., $O(i) = \{k \mid k \in C \cup R, \;\; k \in N_i^-\}$. Similarly, define $O(j) = \{k \mid k \in L \cup C, \;\; k \in N_j^-\}$. Since $L \cup C \not\Rightarrow R$ and $C \cup R \not\Rightarrow L$, we have $|O(i)| \leq f$ and $|O(j)| \leq f$.

Consider the scenario where (i) each node in L has input 0; (ii) each node in R has input ϵ; (iii) nodes in C (if non-empty) have arbitrary inputs in $[0, \epsilon]$; (iv) no node crashes; and (v) the message delay for communications channels from $O(i)$ to i and from $O(j)$ to j is arbitrarily large compared to all the other channels. Recall that such a scenario is possible, since we have assumed that the input range is $[0, K]$, where $K \geq \epsilon$.

Consider node i. Since messages from the set $O(i)$ take arbitrarily long to arrive at node i, and $|O(i)| \leq f$, from the perspective of node i, the nodes in $O(i)$ appear to have crashed. Thus, node i must decide on their output without waiting to hear from the nodes in $O(i)$. Consequently, to satisfy the validity property, the output at node i has to be 0, since 0 is the input of all the nodes in L (from which node i may have received some messages in the execution). Similarly, node j must decide their output without hearing from the nodes in $O(j)$; they must choose output as ϵ, because the input at all the nodes in R is ϵ (from which node j may have received some messages). Thus, the ϵ-agreement property is violated, since the difference between outputs at fault-free nodes i and j is not $< \epsilon$. This is a contradiction. □

Theorem 16 shows that *Condition ICCA* is necessary. Below, we state an equivalent condition *Condition ICCA2* that captures the notion of propagating values. To facilitate the statement, we introduce the notion of "reduced graph", which is based on the notion of graph decomposition and source component introduced in Chapter 3.6.4. It is also similar to the reduced graph introduced in Chapter 4.3. However, the following graph removes exactly f incoming links for each node i; whereas, in Definition 11, we also need to remove up to f nodes before removing links at each remaining nodes.

Definition 21 (Reduced Graph) *For a given graph $G(\mathcal{V}, \mathcal{E})$, a graph $G_f = (\mathcal{V}, \mathcal{E}_f)$ is said to be a* reduced graph*, if $\mathcal{E}_f$ is obtained as follows: at each node, removing f incoming links in $\mathcal{E}$.*

Note that for a given $G(\mathcal{V}, \mathcal{E})$, multiple reduced graphs G_f may exist. Now, we state *Condition ICCA2* based on the concept of reduce graphs:

Condition ICCA2: Consider graph $G(\mathcal{V}, \mathcal{E})$. Every reduced graph G_f obtained as per Definition 21 must contain exactly one *source component.*

Now, we present a key lemma below. The proof is similar to the ones presented in Chapters 5 and 6, and is presented in Appendix B.1.

Lemma 47 Condition ICCA *is equivalent to* Condition ICCA2*.*

Useful Properties Suppose $G(\mathcal{V}, \mathcal{E})$ satisfies *Condition ICCA* and thus *Condition ICCA2* due to Lemma 47. We provide two lemmas below to state some properties of $G(\mathcal{V}, \mathcal{E})$ that are useful for analyzing the iterative algorithm presented later.

Lemma 48 *Suppose that graph $G(\mathcal{V}, \mathcal{E})$ satisfies* Condition ICCA2*. Then, in any reduced graph $G_f = (\mathcal{V}, \mathcal{E}_f)$, there exists a node that has a directed path to all the other nodes in $\mathcal{V}$.*

Proof: Recall that *Condition ICCA2* states that any reduced graph $G_f(\mathcal{V}, \mathcal{E}_f)$ has a single source component. By the definition of source component, any node in the source component (say node s) has directed paths using edges in $\mathcal{E}_f$ to all the other nodes in the source component, since the source component is a strongly connected component. Also, by the uniqueness of the source component, all other strongly connected components in G_f (if any exist) are not source components, and hence reachable from the source component using the edges in $\mathcal{E}_f$. Therefore, node s also has directed paths to all the nodes in $\mathcal{V}$ that are not in the source component as well. Therefore, node s has directed paths to all the other nodes in $\mathcal{V}$ in G_f. This proves the lemma. □

Lemma 49 *For $f > 0$, if graph $G(\mathcal{V}, \mathcal{E})$ satisfies* Condition ICCA*, then each node in $\mathcal{V}$ has in-degree at least $f+1$, i.e., for each $i \in \mathcal{V}$, $|N_i^-| \geq f+1$.*

Proof: The proof is by contradiction. By assumption in the lemma, $f > 0$, and graph $G(\mathcal{V}, \mathcal{E})$ satisfies *Condition ICCA*.

Suppose that there exists a node $i \in \mathcal{V}$ such that $|N_i^-| \leq f$. Define $L = \{i\}, C = \emptyset$, and $R = \mathcal{V} - \{i\}$. Note that sets L, C, R form a partition of $\mathcal{V}$.

Observe that $f > 0$, and $|L \cup C| = 1$. Thus, there can be at most 1 link from $L \cup C$ to any node in R in $G_f = (\mathcal{V}, \mathcal{E}_f)$. Therefore, $L \cup C \not\Rightarrow R$ in G_f. Since $L = \{i\}$, $C = \emptyset$, and $|N_i^-| \leq f$, $C \cup R \not\Rightarrow L$ in G. Thus, G does not satisfy *Condition ICCA*, a contradiction. □

7.3 Algorithm 4

We will prove that there exists a correct IAC algorithm – particularly Algorithm 4 below – that satisfies the termination, validity and ϵ-agreement properties provided that the graph $G(\mathcal{V}, \mathcal{E})$ satisfies *Condition ICCA2*. This implies that *Condition ICCA* and *Condition ICCA2* ares also sufficient. Algorithm 4 has the iterative structure described in Chapter 4.2. Note that it is different from iterative algorithms that were analyzed in previous Chapters, namely Algorithm 1, 2, and 3 due to the nature of asynchronous message delivery and crash faults.

Algorithm 4

1. *Transmit step:* Transmit current state, namely $v_i[t-1]$, on all outgoing edges (to nodes in N_i^+). The message is tagged by index $t-1$.

2. *Receive step:* Wait until the first $|N_i^-| - f$ messages tagged by index $(t-1)$ are received on the incoming edges (breaking ties arbitrarily). Values received in these messages form vector $r_i[t]$ of size $|N_i^-| - f$.

3. *Update step:* Let $N_i^*[t]$ denote the set of nodes from whom the $(t-1)$-indexed messages were received by node i. Note that as proved in Lemma 49, each node has at least $f+1$ incoming neighbors if $f > 0$. Thus, when $f > 0$, $|N_i^*[t]| \geq 1$.

Define

$$v_i[t] \;=\; Z_i(r_i[t], v_i[t-1]) \;=\; \sum_{j \in \{i\} \cup N_i^*[t]} a_i \, v_j[t-1] \tag{7.2}$$

where

$$a_i = \frac{1}{|N_i^*[t]| + 1} = \frac{1}{|N_i^-| + 1 - f}$$

The "weight" of each term on the right-hand side of (7.2) is a_i. Note that $|N_i^*[t]| = |N_i^-| - f$, and $i \notin N_i^*[t]$ because $(i, i) \notin \mathcal{E}$. Thus, the weights on the right-hand side add to 1. Also, $0 < a_i \leq 1$.

Termination: Each node terminates after completing iteration t_{end}, where t_{end} is a constant defined later in Equation (B.4) in Appendix B.2. The value of t_{end} depends on graph $G(\mathcal{V}, \mathcal{E})$, constants U and μ defined earlier in Chapter 7.1 and parameter ϵ in ϵ-agreement property.

7.4 Sufficiency (Correctness of Algorithm 4)

We will prove that given a graph $G(\mathcal{V}, \mathcal{E})$ satisfying *Condition ICCA2*, Algorithm 4 is correct. Therefore, *Condition ICCA2* and *Condition ICCA* are proved to be sufficient. We borrow the matrix analysis from the work on non-fault-tolerant consensus [38, 9, 34]. The proof below follows the same structure in our prior work on node failures in Chapters 5 and 6, and [87, 86]; however, due to asynchronous message delivery, the analysis has some difference as pointed out later.

In the rest of the section, we assume that $G(\mathcal{V}, \mathcal{F})$ satisfies *Condition ICCA2*. We use transition matrix to represent the *Update* step in Algorithm 4, and show how to use the matrix tools discussed in Chapter 5.5.1 to prove the correctness of Algorithm 4 in $G(\mathcal{V}, \mathcal{F})$.

In the discussion below, we use boldface upper case letters to denote matrices, rows of matrices, and their elements. For instance, $\mathbf{A}$ denotes a matrix, $\mathbf{A}_i$ denotes the i-th row of matrix $\mathbf{A}$, and $\mathbf{A}_{ij}$ denotes the element at the intersection of the i-th row and the j-th column of matrix $\mathbf{A}$.

We now introduce more notations:

- For a *given* execution of the proposed algorithm, let F denote the *actual* set of faulty nodes in that execution. Nodes in F may potentially crash.

- For round $r \geq 0$, let $\mathcal{F}[r]$ denote the set of faulty nodes that have crashed before sending any round r messages. Since the algorithm terminates after round t_{end}, we define for $t > t_{end}$, $\mathcal{F}[t] = \mathcal{F}[t_{end}]$. Note that $\mathcal{F}[r] \subseteq \mathcal{F}[r+1] \subseteq F$.

Transition Matrix Representation of Algorithm 4 We describe how to represent Algorithm 4 using a matrix form, namely transition matrix. Let $v[t] \ \ (t_{end} \geq t \geq 0)$, denote a column vector of length n. In the remaining discussion, we will refer to $v[t]$ as the state of the system at the end of round t. In particular, $v_i[t]$ for $i \in V$ is viewed as the state of node i at the end of round t. We define $v[0]$ as the column vector consisting of the initial states at all nodes. That is, the i-th element of $v[0]$, $v_i[0]$, is the initial state of node i.

We will show that the state evolution of Algorithm 4 can be expressed using matrix form as in (7.3) below, where $\mathbf{M}[t]$ is an $n \times n$ matrix with certain desirable properties. The state $v_k[t]$ of node $k \in \mathcal{F}[t]$ is not meaningful, since node k has crashed before sending any round t messages. However, (7.3) assigns it a value for convenience of analysis. $\mathbf{M}[t]$ is said to be the *transition matrix* for round t.

$$v[t] = \mathbf{M}[t] \ v[t-1], \qquad t_{end} \geq t \geq 1 \tag{7.3}$$

In particular, given an execution of the algorithm, we construct the transition matrix $\mathbf{M}[t]$ for round $t \geq 1$ of that execution using the two rules below (*Rule 1* and *Rule 2*). Elements of row $\mathbf{M}_i[t]$ will determine the state $v_i[t]$ at node i, i.e., $v_i[t] = \mathbf{M}_i[t]v[t-1]$. Note that *Rule 1* applies to nodes in $V - \mathcal{F}[t+1]$. Each node $i \in V - \mathcal{F}[t+1]$ survives at least until the start of round $t+1$, and sends at least one message in round $t+1$. Therefore, its state $v_i[t]$ at the end of round t is of consequence. On the other hand, nodes in $\mathcal{F}[t+1]$ have crashed sometime before sending any messages in round $t+1$ (possibly crashing in previous rounds). Thus, their states at the end of round t are not relevant to the fault-free nodes anymore, and hence *Rule 2* defines the entries of the corresponding rows of $\mathbf{M}[t]$ somewhat arbitrarily.

Construction of Transition Matrix $\mathbf{M}[t]$ for $t_{end} \geq t \geq 1$:

- *Rule 1*: For each node $i \in V - \mathcal{F}[t+1]$, and each $k \in V$: Recall that $N_i^*[t]$ denotes the set of nodes from whom the round t messages were received by node i. By construction, $|N_i^*[t]| = |N_i^-| - f$.

  if $k \in N_i^*[t]$, then

$$\mathbf{M}_{ik}[t] := \frac{1}{|N_i^*[t]| + |} = \frac{1}{|N_i^-| - f + 1} \tag{7.4}$$

  Otherwise,

$$\mathbf{M}_{ik}[t] := 0 \tag{7.5}$$

- *Rule 2*: For each node $j \in \mathcal{F}[t+1]$, and each $k \in V$,

$$\mathbf{M}_{jk}[t] \quad := \quad \frac{1}{n} \tag{7.6}$$

Observe that by design, $\mathbf{M}[t]$ is a row stochastic matrix.

Theorem 17 *For $t \geq 1$, define $v[t] = \mathbf{M}[t]v[t-1]$, with $\mathbf{M}[t]$ as specified above. Then, for $\tau \geq 0$, and for all $i \in V - \mathcal{F}[\tau+1]$, $v_i[\tau]$ equals the state at node i in round τ of Algorithm 4.*

Proof: The proof is by induction on τ. Recall that we defined $v_i[0]$ to be equal to input at node i for all $i \in V - \mathcal{F}[1]$. Thus, the theorem trivially holds for $\tau = 0$.

Now, for some $\tau \geq 0$, and for all $i \in V - \mathcal{F}[\tau+1]$, suppose that $v_i[\tau]$ is the state at node i in round τ of Algorithm 4. Recall that nodes in $V - \mathcal{F}[\tau+2]$ surely survive at least till the end of round $\tau+1$ (by definition of $\mathcal{F}[\tau+2]$). Therefore, in round $\tau+1 \geq 1$, each node $i \in V - \mathcal{F}[\tau+2]$ computes its new state $v_i[\tau+1]$ using equation (7.2) as specified in Algorithm 4. Also, if $j \in N_i^*[\tau+1]$, then node j must have sent round $\tau+1$ message to node i. Also, since j did send a round $\tau+1$ message, $j \in V - \mathcal{F}[\tau+1]$. Thus, by induction hypothesis, $v_j[\tau]$ is the state at node j in round τ of Algorithm 4.

Now observe that, by construction $a_i = 1/|N_i^*[\tau+1]+1|$. Thus, the definition of the matrix elements in (7.4) and (7.5) ensures that $\mathbf{M}_i[\tau+1]v[\tau]$ equals $\sum_{j\in\{i\}\cup N_i^*[\tau+1]} a_i\, v_j[\tau]$. Thus, $v_i[\tau+1]$ defined as $\mathbf{M}_i[\tau+1]v[\tau]$ also equals the state at node i in round $\tau+1$ of Algorithm 4. This holds for all $i \in V - \mathcal{F}[\tau+2]$, completing the induction. □

The above theorem states that, for $t_{end} \geq t \geq 1$, equation (7.3), that is, $v[t] = \mathbf{M}[t]v[t-1]$, correctly characterizes the state of the nodes that have not crashed before the end of round t of Algorithm 4. For nodes that have crashed, their states are not relevant, and could be assigned any arbitrary value for analytical purposes (this is what *Rule 2* above effectively does). By repeated application of the state evolution equation (7.3), we obtain

$$v[t] \quad = \quad \left(\Pi_{\tau=1}^{t}\mathbf{M}[\tau]\right) v[0], \qquad t \geq 1 \tag{7.7}$$

Recall that we adopt the “backward” matrix product convention. Then, (7.7) follows from the observation that $\mathbf{M}[\tau](\mathbf{M}[\tau-1]v[\tau-2]) = (\mathbf{M}[\tau]\mathbf{M}[\tau-1])v[\tau-2]$.

Now we state the key lemma. In particular, Lemma 50 allows us to use results for non-homogeneous Markov chains to prove the correctness of Algorithm 4.

Lemma 50 *For $t \geq 1$, transition matrix $\mathbf{M}[t]$ constructed using the procedure above satisfies the following conditions:*

- $\mathbf{M}[t]$ *is a row stochastic matrix.*
- *There exist N_i^r, a subset of incoming neighbors at node i of size f,*[3] *and a constant β $(0 < \beta \leq 1)$ that depends only on graph $G(\mathcal{V}, \mathcal{E})$ such that for each $i \in \mathcal{V}$, and for all $j \in \{i\} \cup (N_i^- - N_i^r)$,*

$$\boldsymbol{M}_{ij}[t] \geq \beta.$$

Proof:

- Observe that, by construction, for each $i \in V$, the row vector $\mathbf{M}_i[t]$ contains only non-negative elements, which add up to 1. Thus, each row $\mathbf{M}_i[t]$ is a stochastic vector, and hence the matrix $\mathbf{M}[t]$ is row stochastic.
- Observe that for round t by choosing $N_i^r = N_i^- - N_i^*[t]$, $|N_i^r| = f$ due to Step 2 of Algorithm 4. Moreover, by the construction of transition matrix above, $\mathbf{M}_{ij}[t] \geq 1/n$ for all $j \in \{i\} \cup N_i^*$. The second claim in the lemma follows by choosing $\beta = 1/n$.

□

Theorem 18 *Algorithm 4 satisfies the* Termination, Validity*, and* ϵ-agreement *properties.*

Given Lemma 50, the proof of Theorem 18 is similar to the proof of Theorem 15 provided in Chapter 6. For completeness, we include the proof in Appendix B.2.

7.5 Summary

This Chapter considers iterative approximate consensus problem in asynchronous systems with crash faults. In particular, we consider the problem in directed graphs. We prove the *tight* necessary and sufficient condition for the communication graphs to have such IAC algorithms.

[3] Intuitively, for $j \in N_i^r$, edge (j, i) corresponds to the links removed in some reduced graph as per Definition 21. Thus, the superscript r in the notation stands for "removed." Also, N_i^r may be different for each round t. For simplicity, the notation does not explicitly represent this dependence.

Chapter 8

Broadcast Using Certified Propagation Algorithm Under f-local Faults

8.1 Introduction

Previous Chapters studied the consensus problem under f-total fault model, generalized fault model and transient Byzantine link fault model. In this Chapter, we explore a closely related problem – fault-tolerant broadcast problem. The fault model under consideration is *f-local model*, in which at most f Byzantine faults occur in the neighborhood of every *fault-free* node [42], and all the links are assumed to be reliable. We identify the necessary and sufficient condition on the underlying communication network topology for the correctness of the Certified Propagation Algorithm (CPA) – the CPA algorithm has been analyzed in prior work [42, 11, 61, 37, 57]. We first study the problem in synchronous systems, and later extend it to asynchronous systems. The results presented in this Chapter are published in [75].

Problem Formulation Consider an arbitrary *directed* network of n nodes, modeled as a directed graph. One node in the network, called the *source* (s), is given an initial input, which the source node needs to transmit to all the other nodes. The source s is assumed to be *fault-free*. We say that CPA is *correct*, if it satisfies the following properties, where x_s denotes the input at source node s:

- **Termination:** every fault-free node i eventually decides on an output value y_i.
- **Validity:** for every fault-free node i, its output value y_i equals the source's input, i.e., $y_i = x_s$.

We study the condition on the network topology for the correctness of CPA. In addition to the related work discussed in Chapter 2, similar condition under other contexts are also discovered by other researchers [70, 30]. [70] proved a similar condition to be sufficient (but not tight) to solve Shamir's (n, k) threshold secret sharing problem, where the source wants to transmit shares of secret to all the other nodes, and all nodes are assumed to be honest-but-curious. In the context of cascading behavior in the network, [30] showed that a similar condition is necessary and sufficient to achieve a complete cascade, i.e., all nodes have learned the value transmitted by a cluster of sources with same input values using only local information.

In their model, all nodes are assumed to be fault-free. Due to our assumption of existence of Byzantine failures, the proofs in this Chapter are different from the ones in [70, 30].

System and Fault Model We consider the synchronous system model and the point-to-point communication network as described in Chapter 1.2 (with the exception of Chapter 8.4.2). We consider the f-local fault model, with at most f incoming neighbors of any fault-free node becoming faulty. As discussed in Chapter 2, [42, 11, 61, 37, 57] also explored this fault model. Yet, to the best of our knowledge, the tight necessary and sufficient conditions for the correctness of CPA in *directed* networks under f-local fault model have not been developed previously.

8.2 Feasibility of CPA under f-local fault model

Certified Propagation Algorithm (CPA) We first describe the Certified Propagation Algorithm (CPA) from [42] formally. Note that the faulty nodes may deviate from this specification arbitrarily. Possible misbehavior includes sending incorrect and mismatching messages to different outgoing neighbors.

Source node s commits to its input x_s at the start of the algorithm, i.e., sets its output equal to x_s. The source node is said to have committed to x_s in round 0. The algorithm for each round r ($r > 0$), is as follows:

1. Each node that commits in round $r-1$ to some value x, transmits message x to all its outgoing neighbors, and then terminates.

2. If any node receives message x directly from source s, it commits to output x.

3. Through round r, if a node has received messages containing value x from at least $f+1$ distinct incoming neighbors, then it commits to output x.

The Necessary Condition For CPA to be correct, the network graph $G(\mathcal{V}, \mathcal{E})$ must satisfy the necessary condition proved in this section. Recall that in Chapter 4.3, we have introduced relations $\Rightarrow$ and $\not\Rightarrow$ in Definition 10. This Chapter will use these relations to define the tight condition.

Definition 22 *Set $F \subseteq \mathcal{V}$ is said to be a feasible f-local fault set, if for each node $v \notin F$, F contains at most f incoming neighbors of node v. That is, for every $v \in \mathcal{V} - F, |N_v^- \cap F| \leq f$.*

We now derive the necessary condition on the network topology.

Theorem 19 *Suppose that CPA is correct in graph $G(\mathcal{V}, \mathcal{E})$ under the f-local fault model. Let sets F, L, R form a partition of $\mathcal{V}$, such that (i) source $s \in L$, (ii) R is non-empty, and (iii) F is a feasible f-local fault set. Then*

- *$L \Rightarrow R$, or*
- *R contains an outgoing neighbor of s, i.e., $N_s^+ \cap R \neq \emptyset$.*

Proof: The proof is by contradiction. Consider any partition F, L, R such that $s \in L$, R is non-empty, and F is a feasible f-local fault set. Suppose that the input at s is x_s. Consider any single execution of the CPA algorithm such that the nodes in F behave as if they have crashed.

By assumption, CPA is correct in the given network under such a behavior by the faulty nodes. Thus, all the fault-free nodes eventually commit their output to x_s. Let round r ($r > 0$), be the earliest round in which at least one of the nodes in R commits to x_s. Let v be one of the node in R that commits in round r. Such a node v must exist since R is non-empty, and it does not contain source node s. For node v to be able to commit, as per specification of the CPA algorithm, either node v should receive the message x_s directly from the source s, or node v must have $f + 1$ distinct incoming neighbors that have already committed to x_s. By definition of node v, nodes that have committed to x_s prior to v must be outside R; since nodes in F behave as crashed, these $f + 1$ nodes must be in L. Thus, either $(s, v) \in \mathcal{E}$, or node v has at least $f + 1$ distinct incoming neighbors in set L. □

Sufficiency We now show that the condition in Theorem 19 is also sufficient.

Theorem 20 *If $G(\mathcal{V}, \mathcal{E})$ satisfies the condition in Theorem 19, then CPA is correct in $G(\mathcal{V}, \mathcal{E})$ under the f-local fault model.*

Proof:

Suppose that $G(\mathcal{V}, \mathcal{E})$ satisfies the condition in Theorem 19. Let F' be the set of faulty nodes. By assumption, F' is a feasible local fault set. Let x_s be the input at source node s. We will show that, (i) fault-free nodes do not commit to any value other than x_s (Validity), and, (ii) until all the fault-free nodes have committed, in each round of CPA, at least one additional fault-free node commits to value x_s (Termination). The proof is by induction.

Induction basis: Source node s commits in round 0 to output equal to its input x_s. No other fault-free nodes commit in round 0.

Induction: Suppose that L is the set of fault-free nodes that have committed to x_s through round r, $r \geq 0$. Thus, $s \in L$. Define $R = \mathcal{V} - L - F'$. If $R = \emptyset$, then the proof is complete. Let us now assume that $R \neq \emptyset$.

Now consider round $r+1$.

- Validity:

  Consider any fault-free node u that has not committed prior to round $r+1$ (i.e., $u \in R$). All the nodes in L have committed to x_s by the end of round r. Thus, in round $r+1$ or earlier, node u may receive messages containing values different from x_s only from nodes in F'. Since there are at most f incoming neighbors of u in F', node u cannot commit to any value different from x_s in round $r+1$.

- Termination:

  By the condition in Theorem 19, there exists a node w in R such that (i) node w has an incoming link from s, or (ii) node w has incoming links from $f+1$ nodes in L. In case (i), node w will commit to x_s on receiving x_s from node s in round $r+1$ (in fact, $r+1$ in this case must be 1). In case (ii), first observe that all the nodes in L from whom node w has incoming links have committed to x_s (by definition of L). Then, node w will be able to commit to x_s after receiving messages from at least $f+1$ incoming neighbors in L, since all nodes in L have committed to x_s by the end of round r by the definition of L.[1] Thus, node w will commit to x_s in round $r+1$.

This completes the proof. □

8.3 CPA without prior knowledge of f

We now present a parameter-independent algorithm CPA-P that does not require prior knowledge of f, and each node only needs to know n, the number of nodes in the system. That is, given a graph G that can tolerate f-local faults (where f is unknown), the algorithm CPA-P presented below solves the broadcast problem in G without usage of f.

The core idea of CPA-P is for each node to exhaustively test all possible parameters by running $n+1$ instances of CPA algorithm in parallel. Each instance of CPA algorithm corresponds to a tested parameter ranging from 0 to n. That is, each instance assumes that the tested parameter is the real bound (f) on the local faults at each node.[2] The correctness of CPA-P is based on the following observation: For each fault-free node, when the tested parameter is larger than or equal to the real parameter f, then there are only two outcomes: (i) it cannot commit, since it did not receive enough identical messages (violating Step

[1] Since node w did not commit prior to round $r+1$, it follows that at least one node in L must have committed in round r.

[2] For simplicity of presentation, we assume that every node keeps track of $n+1$ instances (of the CPA algorithm) at the same time, even if the node already knows that some instances cannot terminate, since it may never receive enough identical messages if the tested parameter is too large. In a real implementation, each node i only needs to keep track of $\lceil \frac{d_i}{2} \rceil - 1$ instances of CPA algorithm, where d_i is the number of incoming neighbors at node i.

3 in CPA as specified in 8.2), or (ii) it commits to a correct value, i.e., the input value of the source. Thus, in the end of the CPA-P,[3] each node can simply commit to the non-null value corresponding to the largest tested parameter. Now, we describe CPA-P formally.

Throughout the execution, each node i (excluding s and outgoing neighbors of s) maintains an $(n+1)$-entry vector v_i, where $v_i[t] \quad (0 \leq t \leq n)$ is the estimate of output corresponding to the tested parameter t. In the beginning of the algorithm, every entry of vector v_i is initialized to be a null value $\perp$, where $\perp$ is distinguished from all possible values of x_s.

Source node s commits to its input x_s at the start of the algorithm (round 0), and transmits message x_s to all its outgoing neighbors in round 1. For the other nodes, the algorithm is as follows.

- For outgoing neighbor of the source s:

  1. In round 1, it receives message x directly from source s, and commits to output x.
  2. In round 2, it transmits messages $< x, 0 >, < x, 1 >, ..., < x, n >$ to all its outgoing neighbors, and terminates.

- For node that is not an outgoing neighbor of s, in each round r $(r > 0)$:

  1. For $0 \leq t \leq n$, each node i that sets $v_i[t]$ in round $r-1$ to some value x, transmits message $< x, t >$ to all its outgoing neighbors.
  2. For $0 \leq t \leq n$, through round r, if a node i has received messages containing value $< x, t >$ from at least $t+1$ distinct incoming neighbors, then it sets $v_i[t] = x$.
  3. In round n, each node i commits to value $v_i[t']$, where t' is the largest value in range $[0, n]$ such that $v_i[t'] \neq \perp$.

  Note that the algorithm performs n rounds.

Now, we show that CPA-P is correct.

Theorem 21 *Given a graph G that can tolerate f-local faults, CPA-P achieves both validity and termination.*

Proof: Denote by CPA-P-t $(0 \leq t \leq n)$ the instance of CPA-P corresponding to the tested parameter t. Then by assumption of G, CPA-P-f is correct. Thus, for each fault-free node i, $v_i[f] = x_s$, the input value at source s. Now, we prove the following claim:

[3] Note that CPA is guaranteed to terminate in n steps, and so is CPA-P.

Claim 7 *For $t > f$, in CPA-P-t, fault-free nodes never decide on an invalid value, i.e., for each fault-free node i, either $v_i[t] = x_s$ or $v_i[t] = \perp$.*

Proof: The proof is by induction.

Induction basis: Source node s and its outgoing neighbors commit to output equal to the source's input x_s in round 0 and 1, respectively. No other fault-free nodes commit in round 0 and 1.

Induction: Suppose that L is the set of fault-free nodes that have committed to x_s through round r $(r > 0)$. Thus, $s \in L$. Let F' be the set of faulty nodes, and $|F'| = f$. Define $R = \mathcal{V} - L - F'$. If $R = \emptyset$, then the proof is complete. Let us now assume that $R \neq \emptyset$.

Now consider round $r + 1$.

Consider any fault-free node u that has not committed prior to round $r+1$ (i.e., $u \in R$). All the nodes in L have committed to x_s by the end of round r. Thus, in round $r+1$ or earlier, node u may receive messages containing values different from x_s only from nodes in F'. Therefore, node u cannot commit to any value different from x_s in round $r + 1$, since by assumption $|N_u^- \cap F'| \leq f < t$.

Unlike the proof in Theorem 20, node u may never gather enough (i.e., at least $t + 1$) identical messages from its incoming neighbors, since $t > f$. Thus, for CPA-P-t, node u may never terminate. In this case, $v_u[t] = \perp$. □

The source node s and fault-free outgoing neighbors of s commit to x_s in round 0 and 1, respectively. By Claim 7 and the fact that CPA-P-f satisfies both validity and termination, each fault-free node i (excluding s and outgoing neighbors of s) commits to x_s. Thus, CPA-P is correct.

□

8.4 Discussion

This section discusses some extensions on the result presented above.

8.4.1 Broadcast Channel

We have so far assumed that the underlying network is a point-to-point network. The results, however, can be easily extended to the *broadcast* or *radio model* [42, 11] as well. In the *broadcast model*, when a node transmits a value, all of its outgoing neighbors receive this value identically. Thus, no node can transmit mismatching values to different outgoing neighbors. Then, it is easy to see that the same condition as the point-to-point network can be shown to be necessary and sufficient for of CPA under the broadcast model as well.

Now consider the following variation of the CPA algorithm: if the outgoing neighbors of source s do not receive a message from s in round 1, the message value is assumed to be some default value. With this modification, the condition in Theorem 19 can also be shown to be necessary and sufficient to perform Byzantine Broadcast [46] under the broadcast model, while satisfying the following three conditions (allowing s to be faulty):

- **Termination:** every fault-free node i eventually decides on an output value y_i.
- **Agreement:** the output values of all the fault-free nodes are equal, i.e., there exists y such that, for every fault-free node $i, y_i = y$.
- **Validity:** if the source node is fault-free, then for every fault-free node i, the output value equals the source's input, i.e., $y = x_s$.

The proof follows from the proof of Theorem 19 and the observation that if s transmits a value, then all the outgoing neighbors of s receive identical value from s, which equals its input x_s when s is fault-free.

8.4.2 Asynchronous Systems

In our analysis so far, we have assumed that the system is synchronous. For a point-to-point network with fault-free source s, it should be easy to see that the condition in Theorem 19 is also necessary and sufficient to achieve agreement using a CPA-like algorithm under the asynchronous systems as well. In this case, the algorithm may not proceed in rounds, but a node still commits to value x either on receiving the value directly from s, or from $f + 1$ nodes. This claim may seem to contradict the FLP result [33]. However, our claim assumes that the source node is fault-free, unlike [33].

8.4.3 Complexity

[57] proved that it is NP-hard to examine whether CPA is correct in a given *undirected* graph with specific f. The condition in [57] is indeed equivalent to our condition (condition in Theorem 19) in *undirected* graphs. Therefore, it is NP-hard to examine whether a given graph satisfies our condition or not.

8.5 Summary

In this Chapter, we consider reliable broadcast problem in arbitrary network using the CPA algorithm in f-local Byzantine fault model. In particular, we provide a *tight* necessary and sufficient condition on the underlying network for the correctness of CPA.

Chapter 9

Convex Hull Consensus under Crash Faults with Incorrect Inputs

9.1 Introduction

The traditional consensus problem formulation assumes that each node has a scalar input, e.g., [5, 52, 60, 29] and the work presented in previous Chapters. As a generalization of this problem, recent work [54, 88, 87] has addressed *vector* consensus (also called multidimensional consensus) in the presence of Byzantine faults, wherein each node has a d-dimensional vector of reals as input, and the nodes reach consensus on a d-dimensional vector within the convex hull of the inputs at fault-free nodes ($d \geq 1$). In the discussion below, it will be more convenient to view a d-dimensional vector as a *point* in the d-dimensional Euclidean space.

This Chapter defines the problem of *convex hull consensus*. Similar to vector consensus, the input at each node is a point in the d-dimensional Euclidean space. However, for convex hull consensus, the output at each node is a *convex polytope* contained within the convex hull of the inputs at the fault-free nodes. Intuitively, the goal is to reach consensus on the "largest possible" polytope within the convex hull of the inputs at fault-free nodes, allowing the nodes to estimate the domain of inputs at the fault-free nodes. In some cases, the output convex polytope may consist of just a single point, but in general, it may contain an infinite number of points in the d-dimensional space. In asynchronous systems, we present an approximate convex hull consensus algorithm with optimal fault tolerance that reaches consensus on optimal output polytope under crash fault model with incorrect inputs (to be defined later). The results presented in this Chapter are published in [79].

Convex hull consensus may be used to solve other related problems. For instance, a solution for convex hull consensus trivially yields a solution for vector consensus [54, 88]. More importantly, convex hull consensus can potentially be used to solve other more interesting problems, such as function optimization with the convex hull of the inputs at fault-free nodes as the domain. We will discuss the application of convex hull consensus to function optimization in Chapter 9.8.

We first describe our fault and system models, and then formally define the convex hull consensus problem.

9.1.1 Models

Fault Model The fault model of interest is different from crash or Byzantine fault models in previous Chapters. This Chapter assumes the *crash faults with incorrect inputs* fault model [20, 5]. In this model, each faulty node have an *incorrect input*, and may crash. A faulty node performs the algorithm faithfully, using an incorrect input, until it (possibly) crashes. The implication of an incorrect input will be clearer when we formally define convex hull consensus below. Roughly speaking, the fault model sits between crash and Byzantine fault models. The faulty node have arbitrary behavior only in the beginning of the algorithm (choosing arbitrarily incorrect inputs); once the algorithm starts, the faulty behavior is the same as in crash fault model studied in previous Chapters. Such faulty behavior is motivated by the existence of malfunctioned sensors.

We assume that at most f nodes may be faulty (f-total fault model), and all fault-free nodes have *correct inputs*. Since this model assumes incorrect inputs at faulty nodes, the simulation techniques in [20, 5] can be used to transform an algorithm designed for this fault model to an algorithm for tolerating Byzantine faults. The transformation requires $n \geq 3f + 1$, which is also a lower bound to solve convex hull consensus under the fault model. (A different Byzantine convex hull consensus algorithm is also briefly discussed in Chapter 9.7 and presented in our technical report [77].) Our results extend naturally to the more commonly used crash fault model wherein faulty nodes also have correct inputs (we will refer to the latter model as crash faults with correct inputs). The extension is presented in Chapter 9.6.

System Model The system under consideration is *asynchronous*, and consists of n nodes. Let the set of nodes be denoted as $V = \{1, 2, \cdots, n\}$. Unlike previous Chapters, this Chapter assumes that all nodes can communicate with each other. Thus, the underlying communication network is modeled as a *complete graph*. Similar to prior work (e.g., [29, 20, 88]), we assume that communication channels are reliable and FIFO (first-in first-out). Each message is delivered exactly once on each channel. The input at node i, denoted as x_i, is a point in the d-dimensional Euclidean space (equivalently, a d-dimensional vector of real numbers).

9.1.2 Convex Hull Consensus

The FLP impossibility of reaching exact consensus in asynchronous systems with crash faults [33] extends to the problem of convex hull consensus as well. Therefore, we consider approximate convex hull consensus. An approximate convex hull consensus algorithm must satisfy the following properties:

- **Validity**: The *output* (or *decision*) at each fault-free node must be a convex polytope in the convex hull of correct inputs. Note that under the crash fault with incorrect inputs model, the input at any faulty node is incorrect.

- **ϵ-Agreement**: For a given constant $\epsilon > 0$, the Hausdorff distance (defined below) between the output polytopes at any two fault-free nodes must be at most ϵ.

- **Termination:** Each fault-free node must terminate within a finite amount of time.

Moreover, the goal of the algorithm is to maximize the size of the output convex polytope at each fault-free node.

Distance Metrics

- $\mathbf{d}_E(p, q)$ denotes the Euclidean distance between points p and q. All points and polytopes in our discussion belong to a d-dimensional Euclidean space, for some $d \geq 1$, even if this is not always stated explicitly.

- For two convex polytopes h_1, h_2, the *Hausdorff distance* $\mathbf{d}_H(h_1, h_2)$ is defined as follows [36].

$$\begin{aligned} \mathbf{d}_H(h_1, h_2) \;=\; \max\ \{ & \max_{p_1 \in h_1} \ \min_{p_2 \in h_2} \mathbf{d}_E(p_1, p_2), \\ & \max_{p_2 \in h_2} \ \min_{p_1 \in h_1} \mathbf{d}_E(p_1, p_2)\} \end{aligned} \tag{9.1}$$

Optimality of Approximate Convex Hull Consensus The algorithm proposed in this paper is optimal in two ways. It requires an optimal number of nodes to tolerate f faults, and it decides on a convex polytope that is optimal in a "worst-case sense", as elaborated below:

- Prior work on approximate vector consensus mentioned earlier [54, 88] showed that $n \geq (d+2)f+1$ is necessary to solve that problem in an asynchronous system consisting of n nodes with at most f Byzantine faults. Although these prior papers dealt with Byzantine faults, it turns out that their proof of lower bound on n (i.e., lower bound of $(d+2)f+1$) is also directly applicable to approximate vector consensus under the crash fault with incorrect inputs model used in our present work. Thus, $n \geq (d+2)f+1$ is a necessary condition for vector consensus under this fault model. Secondly, it is easy to show that an algorithm for approximate convex hull consensus can be transformed into an algorithm for approximate vector consensus. Therefore, $n \geq (d+2)f+1$ is a necessary condition for approximate convex hull consensus as well. Thus, our subsequent discussion under the crash faults with incorrect inputs model assumes that

$$n \geq (d+2)f+1 \tag{9.2}$$

Our algorithm is correct under this condition, and thus achieves optimal fault resilience. For crash faults with correct inputs, a smaller n suffices, as discussed later in Chapter 9.6.

- In this Chapter, we only consider deterministic algorithms. A convex hull consensus algorithm A is said to be optimal if the following condition is true:

  > Let F denote a set of up to f faulty nodes. For a *given execution* of algorithm A with F being the set of faulty nodes, let $y_i(A)$ denote the output polytope at node i at the end of the given execution. For any other convex hull consensus algorithm B, *there exists* an execution with F being the set of faulty nodes, such that $y_i(B)$ is the output at fault-free node i, and $y_j(B) \subseteq y_j(A)$ for *each* fault-free node j.

  The goal here is to decide on an output polytope that includes as much of the convex hull of *all* correct inputs as possible. However, since any node may be potentially faulty (with incorrect input), the output polytope can be smaller than the convex hull of all correct inputs. Intuitively speaking, the optimality condition says that an optimal algorithm should decide on a convex region that is *no smaller than that decided in a worst-case execution* of algorithm B. In Chapter 9.5, we will show that our proposed algorithm is optimal in the above sense.

This Chapter uses similar proof structures in previous Chapters on iterative algorithms. With regards to the proof technique, we show how the above proof structure can be extended to the case when the node state consists of convex polytopes in this Chapter.

9.2 Preliminaries

Here, we introduce functions $\mathcal{H}$, $\mathbf{L}$, and a communication primitive used in our algorithm.

Definition 23 *For a multiset of points X, $\mathcal{H}(X)$ is the convex hull of the points in X.*

A multiset contain the same element more than once.

Definition 24 *Function* $\mathbf{L}$*: Suppose that* ν *non-empty convex polytopes* $h_1, h_2, \cdots, h_\nu$*, and* ν *weights* $c_1, c_2, \cdots, c_\nu$ *are given such that* $0 \le c_i \le 1$ *and* $\sum_{i=1}^{\nu} c_i = 1$*.* Linear combination *of these convex polytopes*

$$\mathbf{L}([h_1, h_2, \cdots, h_\nu] \ ; \ [c_1, c_2, \cdots, c_\nu])$$

is defined as follows:

$p \in \mathbf{L}([h_1, h_2, \cdots, h_\nu]; [\, c_1, c_2, \cdots, c_\nu])$ *if and only if for* $1 \le i \le \nu$*, there exists*

$$p_i \in h_i, \quad \text{such that} \ \ p = \sum_{1 \le i \le \nu} c_i p_i \tag{9.3}$$

Because h_i's above are all convex and non-empty, $\mathbf{L}([h_1, h_2, \cdots, h_\nu] \ ; \ [c_1, c_2, \cdots, c_\nu])$ is also a convex non-empty polytope. (The proof is straightforward.) The parameters for $\mathbf{L}$ consist of two vectors, with the elements of the first vector being polytopes, and the elements of the second vector being the corresponding weights in the linear combination. With a slight abuse of notation, we will also specify the vector of polytopes as a multiset – in such cases, we will always assign an identical weight to all the polytopes in the multiset, and hence their ordering is not important.

***Stable vector* communication primitive** As seen later, our algorithm proceeds in asynchronous rounds. In round 0 of the algorithm, the nodes use a communication primitive called *stable vector* [4, 35], to try to learn each other's inputs. Stable vector was originally developed in the context of crash faults [4] and was later applied to solve Byzantine Barycentric agreement [35]. To achieve its desirable properties (listed below), stable vector requires at least $2f + 1$ nodes when each node either follows the algorithm faithfully or crashes. Since in our crash fault with incorrect inputs model, each node follows the algorithm unless it crashes, the properties of *stable vector* listed below will hold in our context, provided that $n \ge 2f + 1$. As noted earlier in Chapter 9.1, $n \ge (d+2)f+1$ is a necessary condition for approximate convex hull consensus in the presence of crash faults with incorrect inputs. Then, with $d \ge 1$, we have $n \ge 3f + 1$, and the properties of stable vector below will hold.

In round 0 of our algorithm, each node i first broadcasts a message consisting of the tuple $(x_i, i, 0)$, where x_i is node i's input. In this tuple, 0 indicates the (asynchronous) round index. Node i then waits for the stable vector primitive to return a set R_i containing round 0 messages. We will rely on the following properties of the stable vector primitive, which are implied by results proved in prior work [4, 35].

- **Liveness**: At each node i that does not crash before the end of round 0, stable vector returns a set

R_i containing at least $n - f$ distinct tuples of the form $(x, k, 0)$.

- **Containment**: For nodes i, j that do not crash before the end of round 0, let R_i, R_j be the set of messages returned to nodes i, j by stable vector in round 0, respectively. Then, either $R_i \subseteq R_j$ or $R_j \subseteq R_i$. (Also, by the previous property, $|R_i| \geq n - f$ and $|R_j| \geq n - f$.)

Please refer to [4, 35] for the implementation of the *stable vector* primitive.

9.3 *Algorithm CC*

The proposed algorithm, named *Algorithm CC*, proceeds in asynchronous rounds. The input at each node i is named x_i. The initial round of the algorithm is called round 0. Subsequent rounds are named round 1, 2, 3, etc. In each round $t \geq 0$, each node i computes a state variable h_i, which represents a convex polytope in the d-dimensional Euclidean space. We will refer to the value of h_i at the *end* of the t-th round performed by node i as $h_i[t]$, $t \geq 0$. Thus, for $t \geq 1$, $h_i[t-1]$ is the value of h_i at the *start* of the t-th round at node i. The algorithm terminates after t_{end} rounds, where t_{end} is a constant defined later in equation (9.31). The state $h_i[t_{end}]$ of each fault-free node i at the end of t_{end} rounds is its output (or decision) for the consensus algorithm.

X_i and $Y_i[t]$ defined on lines 4 and 13 of the algorithm below are both multisets. A given value may occur multiple times in a multiset. Also, the intersection in line 5 is over the convex hulls of the subsets of multiset X_i of size $|X_i| - f$ (note that each of these subsets is also a multiset). Elements of X_i are points in the d-dimensional Euclidean space, whereas elements of $Y_i[t]$ are convex polytopes. In line 14, $Y_i[t]$ specifies the multiset of polytopes whose linear combination is obtained using **L**; all the weights specified as parameters to **L** here are equal to $1/|Y_i[t]|$

Algorithm CC: Steps performed at node i shown below.

Initialization: All sets used below are initialized to $\emptyset$.

Round 0 **at node** i**:**

```
• On entering round 0:                                                1
      Send $(x_i, i, 0)$ to all the nodes                             2
• When stable vector returns a set $R_i$:                             3
      Multiset $X_i := \{\, x \mid (x, k, 0) \in R_i \}$              4
```

```
        $h_i[0] := \cap_{C \subseteq X_i, |C|=|X_i|-f} \mathcal{H}(C)$                    5
        Proceed to Round 1                                                  6

Round $t \geq 1$ at node $i$:

  • On entering round $t \geq 1$:                                           7
        $\texttt{MSG}_i[t] := \texttt{MSG}_i[t] \cup (h_i[t-1], i, t)$              8
        Send $(h_i[t-1], i, t)$ to all other nodes                          9

  • When message $(h, j, t)$ is received from $j \neq i$                    10
        $\texttt{MSG}_i[t] := \texttt{MSG}_i[t] \cup \{(h, j, t)\}$                 11

  • When $|\texttt{MSG}_i[t]| \geq n - f$ for the first time:                  12
        Multiset $Y_i[t] := \{h \mid (h, j, t) \in \texttt{MSG}_i[t]\}$          13
        $h_i[t] := \mathbf{L}( Y_i[t] ; [\frac{1}{|Y_i[t]|}, \cdots, \frac{1}{|Y_i[t]|}])$          14
        If $t < t_{end}$, then proceed to Round $t+1$                      15
```

Note that stable vector is only used in Round 0. As will be seen later in Chapter 9.5, to achieve optimality of the size of the output polytope, it is important for the intersection of multiset X_i (at line 4) at each fault-free node i to be as large as possible. This property is ensured by receiving messages using stable vector. In later rounds, the goal is to achieve convergence, and in this case, exchanging messages over the reliable channels is enough.

9.4 Correctness of *Algorithm CC*

We prove that *Algorithm CC* satisfies Validity, ϵ-Agreement, and Termination properties. The use of matrix representation in our correctness proof below is inspired by the prior work on non-fault-tolerant consensus (e.g., [38, 9]). We have also used such a proof structure in Chapters 5, 6, and 7, and our prior work on Byzantine consensus [87, 86]. The main differences in this proof are: (i) the state at each node is now a convex polytope (instead of a point), and (ii) the multiplication operation on a vector of convex polytopes and a vector of scalar is now defined using function $\mathbf{L}$ (introduced in Chapter 9.2).

We now introduce more notations and two lemmas:

- For a *given* execution of the proposed algorithm, let F denote the *actual* set of faulty nodes in that execution. Nodes in F have incorrect inputs, and they may potentially crash.

- For round $r \geq 0$, let $\mathcal{F}[r]$ denote the set of faulty nodes that have crashed before sending any round r messages. Since the algorithm terminates after round t_{end}, we define for $t > t_{end}$, $\mathcal{F}[t] = \mathcal{F}[t_{end}]$. Note that $\mathcal{F}[r] \subseteq \mathcal{F}[r+1] \subseteq F$.

Lemma 51 *Algorithm CC ensures* progress*: (i) all the fault-free nodes will eventually progress to round 1; and, (ii) if all the fault-free nodes progress to the start of round t ($t_{end} \geq t \geq 1$), then all the fault-free nodes will eventually progress to the start of round $t+1$.*

Proof:

Part (i): By assumption, all fault-free nodes begin the round 0 eventually, and perform a broadcast of their input (line 1). There at least $3f+1$ nodes as argued in Chapter 9.2, and at most f may crash. The Liveness property of *stable vector* ensures that it will eventually return (on line 3). Therefore, each node that does not crash in round 0 will eventually proceed to round 1 (line 6).

Part (ii): The proof is by induction. Suppose that the fault-free nodes begin round $t \geq 1$. (We already proved that the fault-free nodes begin round 1.) Thus, each fault-free node i will perform a broadcast of $(h_i[t-1], i, t)$ on line 9. By the assumption of reliable channels, node i will eventually receive message $(h_j[t-1], j, t)$ from each fault-free node j. Thus, it will receive messages from at least $n-f-1$ *other* nodes, and include these received messages in $\texttt{MSG}_i[t]$ (line 10-11). Also, it includes (on line 8) its own message into $\texttt{MSG}_i[t]$. Thus, $\texttt{MSG}_i[t]$ is sure to reach size $n-f$ eventually, and node i will be able to progress to round $t+1$ (line 12-15). □

Lemma 52 *For each node $i \in V - \mathcal{F}[1]$, the polytope $h_i[0]$ is non-empty and convex.*

The proof of Lemma 52 uses the following theorem by Tverberg [62]:

Theorem 22 *(Tverberg's Theorem [62]) For any integer $f \geq 0$, for every multiset T containing at least $(d+1)f+1$ points in a d-dimensional space, there exists a partition $T_1, .., T_{f+1}$ of T into $f+1$ non-empty multisets such that $\cap_{l=1}^{f+1} \mathcal{H}(T_l) \neq \emptyset$.*

Proof of Lemma 52 Below, we use Theorem 22 to prove Lemma 52.

Proof: Consider any $i \in V - \mathcal{F}[1]$. Consider the computation of polytope $h_i[0]$ on line 5 of the algorithm as

$$h_i[0] \ := \ \cap_{C \subseteq X_i,\, |C| = |X_i| - f} \quad \mathcal{H}(C), \tag{9.4}$$

where $X_i := \{\, x \,|\, (x, k, 0) \in R_i\}$ (lines 4-5). Convexity of $h_i[0]$ follows directly from (9.4), because $h_i[0]$ is an intersection of convex hulls.

Recall that, due to the lower bound on n discussed in Chapter 9.1, we assume that $n \geq (d+2)f+1$. Thus, $|X_i| \geq n - f \geq (d+1)f+1$. By Theorem 22 above, there exists a partition $T_1, T_2, \cdots, T_{f+1}$ of X_i into multisets (T_j's) such that $\cap_{j=1}^{f+1}\mathcal{H}(T_j) \neq \emptyset$. Let us define

$$J = \cap_{j=1}^{f+1}\mathcal{H}(T_j) \tag{9.5}$$

Thus, by Tverberg's theorem above, J is non-empty. Now, each multiset C used in (9.4) to compute $h_i[0]$ excludes only f elements of X_i, whereas there are $f+1$ multisets in the partition $T_1, T_2, \cdots, T_{f+1}$ of multiset X_i. Hence, each multiset C will fully contain at least one multiset from the partition. It follows that $\mathcal{H}(C)$ will contain J defined above. Since this property holds true for each multiset C used to compute $h_i[0]$, J is contained in the convex polytope $h_i[0]$ computed as per (9.4). Since J is non-empty, $h_i[0]$ is non-empty. □

9.4.1 Matrix Preliminaries

We now introduce some matrix notation and terminology to be used in our proof. Boldface upper case letters are used below to denote matrices, rows of matrices, and their elements. For instance, $\mathbf{A}$ denotes a matrix, $\mathbf{A}_i$ denotes the i-th row of matrix $\mathbf{A}$, and $\mathbf{A}_{ij}$ denotes the element at the intersection of the i-th row and the j-th column of matrix $\mathbf{A}$. A vector is said to be *stochastic* if all its elements are non-negative, and the elements add up to 1. A matrix is said to be row stochastic if each row of the matrix is a stochastic vector [38]. For matrix products, we adopt the "backward" product convention below, where $a \leq b$,

$$\Pi_{\tau=a}^{b}\mathbf{A}[\tau] = \mathbf{A}[b]\mathbf{A}[b-1]\cdots\mathbf{A}[a] \tag{9.6}$$

Let $\mathbf{v}$ be a column vector of size n whose elements are convex polytopes. The i-th element of $\mathbf{v}$ is $\mathbf{v}_i$. Let $\mathbf{A}$ be a $n \times n$ row stochastic square matrix. We define the product of $\mathbf{A}_i$ (the i-th row of $\mathbf{A}$) and $\mathbf{v}$ as follows using function $\mathbf{L}$ defined in Chapter 9.2.

$$\mathbf{A}_i\mathbf{v} \quad = \quad \mathbf{L}(\mathbf{v}^T \,;\, \mathbf{A}_i) \tag{9.7}$$

where ^{T} denotes the transpose operation. The above product is a polytope in the d-dimensional Euclidean space. Product of matrix $\mathbf{A}$ and $\mathbf{v}$ is then defined as follows:

$$\mathbf{Av} = [\ \mathbf{A}_1\mathbf{v} \quad \mathbf{A}_2\mathbf{v} \quad \cdots \quad \mathbf{A}_n\mathbf{v}\ ]^T \tag{9.8}$$

Due to the transpose operation above, the product $\mathbf{Av}$ is a column vector consisting of n polytopes. Now, we present a useful lemma.

Lemma 53 *For two n-by-n matrices $\mathbf{A}$ and $\mathbf{B}$, and an n-element column vector of d-dimensional polytopes $\mathbf{v}$, we have $\mathbf{A}(\mathbf{Bv}) = (\mathbf{AB})\mathbf{v}$.*

Proof: Let $\mathbf{l} = \mathbf{A}(\mathbf{Bv})$ and $\mathbf{r} = (\mathbf{AB})\mathbf{v}$. To prove the lemma, we show that for $1 \leq k \leq n$, $\mathbf{l}_k = \mathbf{r}_k$.

We first show the following claim for an n-element column vector of d-dimensional points p.

Claim 8 $\mathbf{A}_k(\mathbf{B}p) = (\mathbf{AB})_k p$

Proof:

$$\begin{aligned}\mathbf{A}_k(\mathbf{B}p) &= \sum_{i=1}^{n} \mathbf{A}_{ki} \left(\sum_{j=1}^{n} \mathbf{B}_{ij} p_j \right) \\ &= \sum_{j=1}^{n} \left(\sum_{i=1}^{n} \mathbf{A}_{ki}\mathbf{B}_{ij} \right) p_j \\ &= (\mathbf{AB})_k p\end{aligned}$$

□

Claim 9 *For any integer k such that $1 \leq k \leq n$, a point $p \in \mathbf{r}_k$, then $p \in \mathbf{l}_k$.*

Proof: By the definition of p, there exists a vector of d-dimensional points q such that (i) $(\mathbf{AB})_k q = p$; and (ii) for $1 \leq j \leq n$, $q_j \in \mathbf{v}_j$. Observe that $p = (\mathbf{AB})_k q = \mathbf{A}_k(\mathbf{B}q)$ due to Claim 8. This implies that $p \in \mathbf{l}_k$ due to our definition of matrix operation over polytopes. □

Claim 9 implies that

$$\mathbf{r}_k \subseteq \mathbf{l}_k \tag{9.9}$$

Next, we prove the following claim.

Claim 10 *For any integer k such that $1 \leq k \leq n$, if a point $p \in \mathbf{l}_k$, then $p \in \mathbf{r}_k$.*

Proof: By the definition of p, there exists a vector of d-dimensional points p' such that

- $\mathbf{A}_k p' = p$, i.e., $p = \sum_{j=1}^{n} \mathbf{A}_{kj} p'_j$;

- For $1 \leq j \leq n$, there exists a vector of d-dimensional points p^j such that (i) $p'_j = \mathbf{B}_j p^j$; and (ii) $p_i^j \in \mathbf{v}_i$ for each $1 \leq i \leq n$. Note that condition (i) implies that $p'_j = \sum_{i=1}^{n} \mathbf{B}_{ji} p_i^j$.

To prove the claim, we need to find a vector of d-dimensional points q such that $(\mathbf{AB})_k q = p$ and $q_i \in \mathbf{v}_i$ for each $1 \leq i \leq n$. Define q_i as follows:

$$q_i = \frac{\sum_{j=1}^{n} (\mathbf{A}_{kj}\mathbf{B}_{ji})\, p_i^j}{\sum_{j=1}^{n} \mathbf{A}_{kj}\mathbf{B}_{ji}} \tag{9.10}$$

Since by assumption, each $p_i^j \in \mathbf{v}_i$, $q_i \in \mathbf{v}_i$ as well. Now, we show that $(\mathbf{AB})_k q = p$.

$$\begin{aligned}
(\mathbf{AB})_k q &= \sum_{i=1}^{n} \left(\sum_{j=1}^{n} \mathbf{A}_{kj}\mathbf{B}_{ji} \right) q_i \\
&= \sum_{i=1}^{n} \left(\sum_{j=1}^{n} \mathbf{A}_{kj}\mathbf{B}_{ji} \right) \frac{\sum_{j=1}^{n} (\mathbf{A}_{kj}\mathbf{B}_{ji})\, p_i^j}{\sum_{j=1}^{n} \mathbf{A}_{kj}\mathbf{B}_{ji}} \\
&= \sum_{i=1}^{n} \sum_{j=1}^{n} (\mathbf{A}_{kj}\mathbf{B}_{ji})\, p_i^j \\
&= \sum_{j=1}^{n} \mathbf{A}_{kj} \left(\sum_{i=1}^{n} \mathbf{B}_{ji}\ p_i^j \right) \\
&= \sum_{j=1}^{n} \mathbf{A}_{kj} p'_j \\
&= p
\end{aligned}$$

Hence, $p \in \mathbf{r}_k$. □

Claim 10 implies that

$$\mathbf{l}_k \subseteq \mathbf{r}_k \tag{9.11}$$

Equations (9.9) and (9.11) together imply that for each k,

$$\mathbf{l}_k = \mathbf{r}_k$$

Therefore, $\mathbf{l} = \mathbf{r}$. This completes the proof of Lemma 53. □

9.4.2 Algorithm CC in Matrix Form

We describe how to represent Algorithm CC using a matrix form. Let $\mathbf{v}[t]$ $(t_{end} \geq t \geq 0)$, denote a column vector of length n. In the remaining discussion, we will refer to $\mathbf{v}[t]$ as the state of the system at the end of round t. In particular, $\mathbf{v}_i[t]$ for $i \in V$ is viewed as the state of node i at the end of round t. We define $\mathbf{v}[0]$ as follows as *initialization* of the state vector:

(I1) For each node $i \in V - \mathcal{F}[1]$, $\mathbf{v}_i[0] := h_i[0]$. Recall that $h_i[0]$ is the convex hull computed at node i at line 5 in *Algorithm CC*.

(I2) For each node $k \in \mathcal{F}[1]$, first pick any one fault-free node $m \in V - F \subseteq V - \mathcal{F}[1]$, and then $\mathbf{v}_k[0]$ is *arbitrarily* defined to be equal to $h_m[0]$. Such an arbitrary choice suffices because the state $\mathbf{v}_k[0]$ for $k \in \mathcal{F}[1]$ does not impact future state of any other node (by definition, nodes in $\mathcal{F}[1]$ do not send any messages in round 1 and beyond).

We will show that the state evolution of *Algorithm CC* can be expressed using matrix form as in (9.12) below, where $\mathbf{M}[t]$ is an $n \times n$ matrix with certain desirable properties. The state $\mathbf{v}_k[t]$ of node $k \in \mathcal{F}[t]$ is not meaningful, since node k has crashed before sending any round t messages. However, (9.12) assigns it a value for convenience of analysis. $\mathbf{M}[t]$ is said to be the *transition matrix* for round t.

$$\mathbf{v}[t] = \mathbf{M}[t]\ \mathbf{v}[t-1], \qquad t_{end} \geq t \geq 1 \tag{9.12}$$

In particular, given an execution of the algorithm, we construct the transition matrix $\mathbf{M}[t]$ for round $t \geq 1$ of that execution using the two rules below (*Rule 1* and *Rule 2*). Elements of row $\mathbf{M}_i[t]$ will determine the state $\mathbf{v}_i[t]$ of node i (specifically, $\mathbf{v}_i[t] = \mathbf{M}_i[t]\mathbf{v}[t-1]$). Note that *Rule 1* applies to nodes in $V - \mathcal{F}[t+1]$. Each node $i \in V - \mathcal{F}[t+1]$ survives at least until the start of round $t+1$, and sends at least one message in round $t+1$. Therefore, its state $\mathbf{v}_i[t]$ at the end of round t is of consequence. On the other hand, nodes in $\mathcal{F}[t+1]$ crash sometime before sending any messages in round $t+1$ (possibly crashing in previous rounds). Thus, their states at the end of round t are not relevant to the fault-free nodes anymore, and hence *Rule 2* defines the entries of the corresponding rows of $\mathbf{M}[t]$ somewhat arbitrarily. Note that the construction is similar to the one presented in Chapter 7.

Construction of Transition Matrix $\mathbf{M}[t]$ for $t_{end} \geq t \geq 1$:

In the matrix specification below, $\mathtt{MSG}_i[t]$ is the message set at the point where $Y_i[t]$ is defined on line 13 of the algorithm. Thus, $Y_i[t] := \{h \mid (h, j, t) \in \mathtt{MSG}_i[t]\}$, and $|\mathtt{MSG}_i[t]| = |Y_i[t]|$.

- *Rule 1*: For each node $i \in V - \mathcal{F}[t+1]$, and each $k \in V$:

If a round t message from node k (of the form $(*, k, t)$) is in $\texttt{MSG}_i[t]$, then

$$\mathbf{M}_{ik}[t] := \frac{1}{|\texttt{MSG}_i[t]|} \tag{9.13}$$

Otherwise,

$$\mathbf{M}_{ik}[t] := 0 \tag{9.14}$$

- *Rule 2*: For each node $j \in \mathcal{F}[t+1]$, and each $k \in V$,

$$\mathbf{M}_{jk}[t] \quad := \quad \frac{1}{n} \tag{9.15}$$

Observe that by design, $\mathbf{M}[t]$ is a row stochastic matrix.

Theorem 23 *For $t \geq 1$, define $\mathbf{v}[t] = \mathbf{M}[t]\mathbf{v}[t-1]$, with $\mathbf{M}[t]$ as specified above. Then, for $\tau \geq 0$, and for all $i \in V - \mathcal{F}[\tau+1]$, $\mathbf{v}_i[\tau]$ equals $h_i[\tau]$.*

The proof is similar to proof of Theorem 17 in Chapter 7 and is presented below for completeness.

Proof: The proof of the above theorem is by induction on τ. Recall that we defined $\mathbf{v}_i[0]$ to be equal to $h_i[0]$ for all $i \in V - \mathcal{F}[1]$ in the initialization step (I1) in Chapter 9.4. Thus, the theorem trivially holds for $\tau = 0$.

Now, for some $\tau \geq 0$, and for all $i \in V - \mathcal{F}[\tau+1]$, suppose that $\mathbf{v}_i[\tau] = h_i[\tau]$. Recall that nodes in $V - \mathcal{F}[\tau+2]$ surely survive at least till the end of round $\tau+1$ (by definition of $\mathcal{F}[\tau+2]$). Therefore, in round $\tau+1 \geq 1$, each node in $i \in V - \mathcal{F}[\tau+2]$ computes its new state $h_i[\tau+1]$ at line 14 of Algorithm CC, using function $\mathbf{L}(\ Y_i[\tau+1]\ ;\ [\frac{1}{|Y_i[\tau+1]|}, \cdots, \frac{1}{|Y_i[\tau+1]|}])$, where $Y_i[\tau+1] := \{h \mid (h, j, \tau+1) \in \texttt{MSG}_i[\tau+1]\}$. Also, if $(h, j, \tau+1) \in \texttt{MSG}_i[\tau+1]$, then node j must have sent round $\tau+1$ message $(h_j[\tau], j, \tau+1)$ to node i – in other words, h above (in $(h, j, \tau+1) \in \texttt{MSG}_i[\tau+1]$) must be equal to $h_j[\tau]$. Also, since j did send a round $\tau+1$ message, $j \in V - \mathcal{F}[\tau+1]$. Thus, by induction hypothesis, $\mathbf{v}_j[\tau] = h_j[\tau]$.

Now observe that, by definition of $Y_i[\tau+1]$ at line 13 of the algorithm, $|Y_i[\tau+1]| = |\texttt{MSG}_i[\tau+1]|$. Thus, the definition of the matrix elements in (9.13) and (9.14) ensures that $\mathbf{M}_i[\tau+1]\mathbf{v}[\tau]$ equals $\mathbf{L}(\ Y_i[\tau+1]\ ;\ [\frac{1}{|Y_i[\tau+1]|}, \cdots, \frac{1}{|Y_i[\tau+1]|}])$, i.e., $h_i[\tau+1]$. Thus, $\mathbf{v}_i[\tau+1]$ defined as $\mathbf{M}_i[\tau+1]\mathbf{v}[\tau]$ also equals $h_i[\tau+1]$. This holds for all $i \in V - \mathcal{F}[\tau+2]$, completing the induction. □

9.4.3 Property of Transition Matrix

To prove Lemma 9.4.3, we first prove the following lemma.

Lemma 54 *For $t \geq 1$, transition matrix $\mathbf{M}[t]$ constructed using the procedure described in Chapter 9.4 satisfies the following conditions:*

- *$\mathbf{M}[t]$ is a row stochastic matrix.*
- *For $i, j \in V - \mathcal{F}[t+1]$, there exists a fault-free node $g(i,j)$ such that $\mathbf{M}_{ig(i,j)}[t] \geq \frac{1}{n}$ and $\mathbf{M}_{jg(i,j)}[t] \geq \frac{1}{n}$*

Proof:

- Observe that, by construction, for each $i \in V$, the row vector $\mathbf{M}_i[t]$ contains only non-negative elements, which add up to 1. Thus, each row $\mathbf{M}_i[t]$ is a stochastic vector, and hence the matrix $\mathbf{M}[t]$ is row stochastic.
- To prove the second claim in the lemma, consider any pair of nodes $i, j \in V - \mathcal{F}[t+1]$. Recall that the set $\texttt{MSG}_i[t]$ used in the construction of $\mathbf{M}[t]$ is such that $|\texttt{MSG}_i[t]| = |Y_i[t]|$ (i.e., $\texttt{MSG}_i[t]$ is the message set at the point where $Y_i[t]$ is created). Thus, $|\texttt{MSG}_i[t]| \geq n - f$ and $|\texttt{MSG}_j[t]| \geq n - f$, and there must be at least $n - 2f$ messages in $\texttt{MSG}_i[t] \cap \texttt{MSG}_j[t]$. By assumption, $n \geq (d+2)f + 1$. Hence, $n - 2f \geq df + 1 \geq f + 1$, since $d \geq 1$. Therefore, there exists a fault-free node $g(i,j)$ such that $(h_{g(i,j)}[t-1], g(i,j), t) \in \texttt{MSG}_i[t] \cap \texttt{MSG}_j[t]$. By (9.13) in the procedure to construct $\mathbf{M}[t]$, $\mathbf{M}_{ig(i,j)}[t] = \frac{1}{|\texttt{MSG}_i[t]|} \geq \frac{1}{n}$ and $\mathbf{M}_{jg(i,j)}[t] = \frac{1}{|\texttt{MSG}_j[t]|} \geq \frac{1}{n}$.

□

The above theorem states that, for $t_{end} \geq t \geq 1$, equation (9.12), that is, $\mathbf{v}[t] = \mathbf{M}[t]\mathbf{v}[t-1]$, correctly characterizes the state of the nodes that have not crashed before the end of round t. For nodes that have crashed, their states are not relevant, and could be assigned any arbitrary value for analytical purposes (this is what *Rule 2* above effectively does). Given the matrix product definition in (9.8), and by Lemma 53 and repeated application of the state evolution equation (9.12), we obtain

$$\mathbf{v}[t] \quad = \quad \left(\Pi_{\tau=1}^{t} \mathbf{M}[\tau]\right) \mathbf{v}[0], \qquad t \geq 1 \tag{9.16}$$

Recall that we adopt the "backward" matrix product convention presented in (9.6). Then, (9.16) follows from the observation that $\mathbf{M}[\tau](\mathbf{M}[\tau-1]\mathbf{v}[\tau-2]) = (\mathbf{M}[\tau]\mathbf{M}[\tau-1])\mathbf{v}[\tau-2]$.

9.4.4 Correctness Proof

Definition 25 *A polytope is* valid *if it is contained in the convex hull of the inputs of fault-free nodes.*

Now, we present three lemmas that are used in the correctness proof below. The following lemma specifies the properties of the multiplication of a series of transition matrices $\mathbf{M}[\tau]$ constructed using the procedure above.

Lemma 55 *For $t \geq 1$, let $\mathbf{P}[t] = \Pi_{\tau=1}^{t} \mathbf{M}[\tau]$. Then,*

- *$\mathbf{P}[t]$ is a row stochastic matrix.*
- *For $i, j \in V - F$, and $k \in V$,*

$$\| \mathbf{P}_{ik}[t] - \mathbf{P}_{jk}[t] \| \leq \left(1 - \frac{1}{n}\right)^{t} \tag{9.17}$$

where $\|a\|$ denotes absolute value of real number a.

The proof is similar to the ones in Chapters 5, 6 or 7 and is presented here fore completeness. Note that proof below uses matrix tools described in Chapter 5.5.1.

Proof: By the first claim of Lemma 54, $\mathbf{M}[\tau]$ for $1 \leq \tau \leq t$ is row stochastic. Thus, $\mathbf{P}[t]$ is a product of row stochastic matrices, and hence, it is itself also row stochastic.

Now, observe that by the second claim in Lemma 54 and Claim 33, $\lambda(\mathbf{M}[t]) \leq 1 - \frac{1}{n} < 1$. Then by Claim 32,

$$\delta(\mathbf{P}[t]) = \delta(\Pi_{\tau=1}^{t} \mathbf{M}[\tau]) \ \leq \ \Pi_{\tau=1}^{t} \lambda(\mathbf{M}[\tau]) \ \leq \ \left(1 - \frac{1}{n}\right)^{t} \tag{9.18}$$

Consider any two fault-free nodes $i, j \in V - F$. By (9.18), $\delta(\mathbf{P}[t]) \leq \left(1 - \frac{1}{n}\right)^{t}$. Therefore, by the definition of $\delta(\cdot)$, for $1 \leq k \leq n$, we have

$$\|\mathbf{P}_{ik}[t] - \mathbf{P}_{jk}[t]\| \leq \left(1 - \frac{1}{n}\right)^{t} \tag{9.19}$$

□

Lemma 56 *$h_i[0]$ for each node $i \in V - \mathcal{F}[1]$ is valid.*

Proof: Recall that $h_i[0]$ is obtained on line 5 of Algorithm CC as

$$h_i[0] \ := \ \cap_{C \subseteq X_i, \, |C| = |X_i| - f} \quad \mathcal{H}(C),$$

where $X_i = \{ x \mid (x, k, 0) \in R_i \}$. Under the *crash faults with incorrect inputs* model, except for up to f values in X_i (which may correspond to inputs at faulty nodes), all the other values in X_i must correspond to inputs at fault-free nodes (and hence they are correct). Therefore, at least one set C used in the computation of

$h_i[0]$ must contain only the inputs at fault-free nodes. Therefore, $h_i[0]$ is in the convex hull of the inputs at fault-free nodes. That is, $h_i[0]$ is valid. □

Lemma 57 *Suppose non-empty convex polytopes $h_1, \cdots, h_\nu$ are all valid. Consider ν constants $c_1, c_2, \cdots, c_\nu$ such that $0 \le c_i \le 1$ and $\sum_{i=1}^{\nu} c_i = 1$. Then the linear combination of these convex polytopes, $\boldsymbol{L}([h_1, h_2, \cdots, h_\nu]\,;\,[c_1, c_2, \cdots, c_\nu])$, is convex, non-empty, and valid.*

Proof: Polytopes $h_1, \cdots, h_\nu$ are given as non-empty, convex, and valid. Let

$$L := \mathbf{L}([h_1, h_2, \cdots, h_\nu]\,;\; [c_1, c_2, \cdots, c_\nu]) \tag{9.20}$$

We will show that L is convex, non-empty, and valid.

*L **is convex:*** Given any two points x, y in L, by Definition 24, we have

$$x = \sum_{1 \le i \le \nu} c_i p_{(i,x)} \quad \text{for some } p_{(i,x)} \in h_i, 1 \le i \le \nu \tag{9.21}$$

and

$$y = \sum_{1 \le i \le \nu} c_i p_{(i,y)} \quad \text{for some } p_{(i,y)} \in h_i, \quad 1 \le i \le \nu \tag{9.22}$$

Now, we show that any convex combination of x and y is also in L defined in (9.20). Consider a point z such that

$$z = \theta x + (1-\theta) y \quad \text{where } 0 \le \theta \le 1 \tag{9.23}$$

Substituting (9.21) and (9.22) into (9.23), we have

$$\begin{aligned} z &= \theta \sum_{1 \le i \le \nu} c_i p_{(i,x)} + (1-\theta) \sum_{1 \le i \le \nu} c_i p_{(i,y)} \\ &= \sum_{1 \le i \le \nu} c_i \left(\theta p_{(i,x)} + (1-\theta) p_{(i,y)}\right) \end{aligned} \tag{9.24}$$

Define $p_{(i,z)} = \theta p_{(i,x)} + (1-\theta) p_{(i,y)}$ for $1 \le i \le \nu$. Since h_i is convex, and $p_{(i,z)}$ is a convex combination of $p_{(i,x)}$ and $p_{(i,y)}$, $p_{(i,z)}$ is also in h_i. Substituting the definition of $p_{(i,z)}$ in (9.24), we have

$$z = \sum_{1 \le i \le \nu} c_i \; p_{(i,z)} \quad \text{where } p_{(i,z)} \in h_i, \quad 1 \le i \le \nu$$

Hence, by Definition 24, z is also in L. Therefore, L is convex.

L is non-empty: The proof that L is non-empty is trivial. Since each of the h_i's is non-empty, there exists at least one point $z_i \in h_i$ for $1 \le i \le \nu$. Then $\sum_{1 \le i \le \nu} c_i z_i$ is in L, and hence L is non-empty.

L is valid: The proof that L is valid is also straightforward. Since each of the h_i's is valid, each point in each h_i is a convex combination of the inputs at the fault-free nodes. Since each point in L is a convex combination of points in h_i's, it then follows that each point in L is in the convex hull of the inputs at fault-free nodes. □

Theorem 24 *Algorithm CC satisfies the* validity, ϵ-agreement *and* termination *properties.*

Proof: We prove that Algorithm CC satisfies the *validity*, *ϵ-agreement* and *termination* properties after a large enough number of asynchronous rounds.

Repeated applications of Lemma 51 ensures that the fault-free nodes will progress from round 0 through round r, for any $r \ge 0$, allowing us to use (9.16). Consider round $t \ge 1$. Let

$$\mathbf{P}[t] \quad = \quad \Pi_{\tau=1}^{t} \mathbf{M}[\tau] \tag{9.25}$$

Validity:

We prove validity using the series of observations below:

- *Observation 1*: By Lemma 52 and Lemma 56, $h_i[0]$ for each $i \in \mathcal{V} - \mathcal{F}[1]$ is non-empty and valid. Also, each such $h_i[0]$ is convex by construction (line 5 of Algorithm CC).

- *Observation 2*: As per the initialization step (I1) in Chapter 9.4, for each $i \in V - \mathcal{F}[1]$, $\mathbf{v}_i[0] := h_i[0]$; thus, by Observation 1 above, for each such node i, $\mathbf{v}_i[0]$ is convex, valid and non-empty. Also, in initialization step (I2), for each node $k \in \mathcal{F}[1]$, we set $\mathbf{v}_k[0] := h_m[0]$, where m is a fault-free node; thus, by Observation 1, for each such node k, $\mathbf{v}_k[0]$ is convex, valid and non-empty. Therefore, each element of $\mathbf{v}[0]$ is a non-empty, convex and valid polytope.

- *Observation 3*: By Lemma 55, $\mathbf{P}[t]$ is a *row stochastic* matrix. Thus, elements of each row of $\mathbf{P}[t]$ are non-negative and add up to 1. Therefore, by Observation 2 above, and Lemma 57, $\mathbf{P}_i[t]\mathbf{v}[0]$ for each $i \in V - F$ is valid, convex and non-empty. Also, by Theorem 23, and equation (9.16), $h_i[t] = \mathbf{P}[t]\mathbf{v}[0]$ for $i \in V - F$. Thus, $h_i[t]$ is valid, convex and non-empty for $t \geq 1$.

Therefore, *Algorithm CC* satisfies the validity property.

ϵ-Agreement and Termination:

Nodes in $\mathcal{F}[1]$ do not send any messages to any other node in round 1 and beyond. Thus, by the construction of $\mathbf{M}[t]$, for each $a \in V - \mathcal{F}[1]$ and $b \in \mathcal{F}[1]$, $\mathbf{M}_{ab}[t] = 0$ for all $t \geq 1$; it then follows that $\mathbf{P}_{ab}[t] = 0$ as well.[1]

Consider fault-free nodes $i, j \in V - F$. The previous paragraph implies that, for any point q_i in $h_i[t] = \mathbf{v}_i[t] = \mathbf{P}_i[t]\mathbf{v}[0]$, there must exist, for all $k \in V - \mathcal{F}[1]$, $p_k \in h_k[0]$, such that

$$q_i = \sum_{k \in V - \mathcal{F}[1]} \mathbf{P}_{ik}[t] p_k \tag{9.26}$$

Using points p_k in the above equation, now choose point q_j in $h_j[t]$ defined as follows.

$$q_j = \sum_{k \in V - \mathcal{F}[1]} \mathbf{P}_{jk}[t] p_k \tag{9.27}$$

For points p_k, denote by $p_k(l)$ the value of p_k's l-th coordinate. Then, (9.26) and (9.27) imply the following equalities for $d \geq l \geq 1$, respectively:

$$q_i(l) = \sum_{k \in V - \mathcal{F}[1]} \mathbf{P}_{ik}[t] p_k(l) \tag{9.28}$$

and

$$q_j(l) = \sum_{k \in V - \mathcal{F}[1]} \mathbf{P}_{jk}[t] p_k(l) \tag{9.29}$$

Recall that the Euclidean distance between q_i and q_j is $\mathbf{d}_E(q_i, q_j)$. From Lemma 55, (9.28) and (9.29), we have the following:

[1] Claim 11 in Chapter 9.5 relates to this observation.

$$\mathbf{d}_E(q_i, q_j) = \sqrt{\sum_{l=1}^{d} (q_i(l) - q_j(l))^2}$$

$$\begin{aligned}
&= \sqrt{\sum_{l=1}^{d} \left(\sum_{k \in V - \mathcal{F}[1]} \mathbf{P}_{ik}[t] p_k(l) - \sum_{k \in V - \mathcal{F}[1]} \mathbf{P}_{jk} p_k(l) \right)^2} \\
&= \sqrt{\sum_{l=1}^{d} \left(\sum_{k \in V - \mathcal{F}[1]} (\mathbf{P}_{ik}[t] - \mathbf{P}_{jk}[t]) p_k(l) \right)^2} \\
&\leq \sqrt{\sum_{l=1}^{d} \left[\left(1 - \frac{1}{n}\right)^{2t} \left(\sum_{k \in V - \mathcal{F}[1]} \|p_k(l)\| \right)^2 \right]} \\
&= \left(1 - \frac{1}{n}\right)^{t} \sqrt{\sum_{l=1}^{d} \left(\sum_{k \in V - \mathcal{F}[1]} \|p_k(l)\| \right)^2}
\end{aligned}$$

The second equality above is due to (9.28) and (9.29), and the fourth inequality is due to Lemma 55.

Define

$$\Omega = \max_{p_k \in h_k[0], k \in V - \mathcal{F}[1]} \sqrt{\sum_{l=1}^{d} (\sum_{k \in V - \mathcal{F}[1]} \|p_k(l)\|)^2}$$

Therefore, $\mathbf{d}_E(q_i, q_j)$ is upper bounded by

$$\left(1 - \frac{1}{n}\right)^{t} \sqrt{\sum_{l=1}^{d} \left(\sum_{k \in V - \mathcal{F}[1]} \|p_k(l)\| \right)^2} \; \leq \; (1 - \frac{1}{n})^t \, \Omega \tag{9.30}$$

Because the $h_k[0]$'s in the definition of Ω are all valid (by Lemma 56), Ω can itself be upper bounded by a function of the input vectors at the fault-free nodes. In particular, under the assumption that each element of fault-free nodes' input vectors is upper bounded by U and lower bounded by μ, Ω is upper bounded by $\sqrt{dn^2 \max(U^2, \mu^2)}$. Observe that the upper bound on the right-hand-side of (9.30) monotonically decreases with t, because $1 - \frac{1}{n} < 1$. Define t_{end} as the smallest positive integer t for which

$$\left(1 - \frac{1}{n}\right)^{t} \sqrt{dn^2 \max(U^2, \mu^2)} < \epsilon \tag{9.31}$$

Recall that the algorithm terminates after t_{end} rounds. Since t_{end} is finite, the algorithms satisfies the *termination* condition.

(9.30) and (9.31) together imply that, for fault-free nodes i, j and for each point $q_i \in h_i[t_{end}]$, there exists a point $q_j[t] \in h_j[t_{end}]$, such that $\mathbf{d}_E(q_i, q_j) < \epsilon$ (and, similarly, vice-versa). Thus, by Definition of Hausdorff distance, $\mathbf{d}_H(h_i[t_{end}], h_j[t_{end}]) < \epsilon$. Since this holds true for any pair of fault-free nodes i, j, the ϵ-agreement property is satisfied at termination. □

Even though we only show that validity and ϵ-agreement properties hold for fault-free nodes, these two properties hold for all nodes that do not crash before completing the algorithm.

9.5 Optimality of *Algorithm CC*

Due to the *Containment* property of stable vector mentioned in Chapter 9.2, the set Z defined below contains at least $n - f$ messages. Recall that set R_i is defined on line 3 of Algorithm CC.

$$Z \quad := \quad \cap_{i \in V - F} \; R_i \tag{9.32}$$

Define multiset $X_Z := \{x \mid (x, k, 0) \in Z\}$. Then, define a convex polytope I_Z as follows.

$$I_Z \quad := \quad \cap_{D \subset X_Z, |D| = |X_Z| - f} \; \mathcal{H}(D) \tag{9.33}$$

Now we establish a "lower bound" on output at the fault-free nodes.

Lemma 58 *For all $i \in V - \mathcal{F}[t+1]$ and $t \geq 0$, $I_Z \subseteq h_i[t]$.*

Proof:

We first present a claim that will be used in the proof of Lemma 58.

Claim 11 *For $t \geq 1$, let $\mathbf{P}[t] = \Pi_{\tau=1}^{t} \mathbf{M}[\tau]$. Then, for all nodes $j \in V - \mathcal{F}[t+1]$, and $k \in \mathcal{F}[1]$, $\mathbf{P}_{jk}[t] = 0$.*

Proof: The claim is intuitively straightforward. For completeness, we present a formal proof here. The proof is by induction on t.

Induction Basis: Consider the case when $t = 1$, $j \in V - \mathcal{F}[2]$, and $k \in \mathcal{F}[1]$. Then by definition of $\mathcal{F}[1]$, $(*, k, 0) \notin \mathtt{MSG}_j[1]$. Then, due to (9.14), $\mathbf{M}_{jk}[1] = 0$, and hence $\mathbf{P}_{jk}[1] = \mathbf{M}_{jk}[1] = 0$.

Induction: Consider $t \geq 2$. Assume that the claim holds true through round $t - 1$. Then, $\mathbf{P}_{jk}[t-1] = 0$ for all $j \in V - \mathcal{F}[t]$ and $k \in \mathcal{F}[1]$. Recall that $\mathbf{P}[t-1] = \Pi_{\tau=1}^{t-1} \mathbf{M}[\tau]$.

Now, we will prove that the claim holds true for round t. Consider $j \in V - \mathcal{F}[t+1]$ and $k \in \mathcal{F}[1]$. Note that $\mathbf{P}[t] = \Pi_{\tau=1}^{t} \mathbf{M}[\tau] = \mathbf{M}[t] \Pi_{\tau=1}^{t-1} \mathbf{M}[\tau] = \mathbf{M}[t]\mathbf{P}[t-1]$. Thus, $\mathbf{P}_{jk}[t]$ can be non-zero only if there exists a $q \in V$ such that $\mathbf{M}_{jq}[t]$ and $\mathbf{P}_{qk}[t-1]$ are both non-zero.

For any $q \in \mathcal{F}[t-1]$, $(*, q, t-1) \notin \mathtt{MSG}_j[t]$. Then, due to (9.14), $\mathbf{M}_{jq}[t] = 0$ for all $q \in \mathcal{F}[t-1]$, and hence all $q \in \mathcal{F}[1]$ (note that $\mathcal{F}[r-1] \subseteq \mathcal{F}[r]$ for $r \geq 2$). Additionally, by the induction hypothesis, for all $q \in V - \mathcal{F}[t]$ and $k \in \mathcal{F}[1]$, $\mathbf{P}_{qk}[t-1] = 0$. Thus, these two observations together imply that there does not exist any $q \in V$ such that $\mathbf{M}_{jq}[t]$ and $\mathbf{P}_{qk}[t-1]$ are both non-zero. Hence, $\mathbf{P}_{jk}[t] = 0$. □

Proof of Lemma 58 Recall that Z and I_Z are defined in (9.32) and (9.33), respectively. We first prove that for all $j \in V - \mathcal{F}[1]$, $I_Z \subseteq h_j[0]$. We make the following observations for each node $i \in V - \mathcal{F}[1]$:

- *Observation 1*: By the definition of multiset X_i at line 4 of round 0 at node i, and the definition of X_Z in Chapter 9.5, we have $X_Z \subseteq X_i$.
- *Observation 2*: Let A and B be sets of points in the d-dimensional space, where $|A| \geq n-f$, $|B| \geq n-f$ and $A \subseteq B$. Define $h_A := \cap_{C_A \subseteq A, |C_A| = |A|-f} \ \mathcal{H}(C_A)$ and $h_B := \cap_{C_B \subseteq B, |C_B| = |B|-f} \ \mathcal{H}(C_B)$. Then $h_A \subseteq h_B$. This observation follows directly from the fact that every multiset C_A in the computation of h_A is contained in some multiset C_B used in the computation of h_B, and the property of function $\mathcal{H}$.

Now, consider the computation of $h_i[0]$ at line 5. By Observations 1 and 2, and the definitions of $h_i[0]$ and I_Z, we have that $I_Z \subseteq h_i[0] = \mathbf{v}_i[0]$, where $i \in V - \mathcal{F}[1]$. Also, by initialization step (I2) (in Chapter 9.4), for $k \in \mathcal{F}[1]$, $\mathbf{v}_k[0] = h_m[0]$, for some fault-free node m. Thus, all the elements of $\mathbf{v}[0]$ contain I_Z. Then, due to row stochasticity of $\Pi_{\tau=1}^{t}\mathbf{M}[\tau]$, it follows that each element of $\mathbf{v}[t] = (\Pi_{\tau=1}^{t}\mathbf{M}[\tau]) \ \mathbf{v}[0]$ also contain I_Z. Recall that $h_i[t] = \mathbf{v}_i[t]$ for each fault-free node. Thus, Lemma 58 is proved. □

The following theorem follows from Lemma 58.

Theorem 25 Algorithm CC *is optimal under the notion of optimality in Chapter 9.1.*

Proof: Consider multiset X_Z defined in Chapter 9.5. Recall that $|X_Z| = |Z|$, and that Z contains at least $n-f$ tuples. Thus, X_Z contains at least $n-f$ points, and of these, at least $n-2f$ points must be the inputs at fault-free nodes. Let V_Z denote the set of fault-free nodes whose inputs appear in X_Z.

Now consider the following execution of any algorithm ALGO that correctly solves approximate convex hull consensus. Suppose that the faulty nodes in F do not crash, and have incorrect inputs. Consider the case when nodes in $V - X_Z$ are so slow that the other fault-free nodes must terminate before receiving any messages from the nodes in $V - X_Z$. This is possible, since we assume that faulty nodes do not crash, and $|V - X_Z| \leq f$ (due to $|X_Z| \geq n-f$). The fault-free nodes in V_Z cannot determine whether the nodes in $V - X_Z$ are just slow, or they have crashed.

Nodes in V_Z must be able to terminate without receiving any messages from the nodes in $V - X_Z$. Thus, their output must be in the convex hull of inputs at the fault-free nodes whose inputs are included in X_Z. However, any f of the nodes whose inputs are in X_Z may potentially be faulty and have incorrect inputs. Therefore, the output obtained by ALGO must be contained in I_Z as defined in Chapter 9.5. On the other hand, by Lemma 58, the output obtained using Algorithm CC contains I_Z. This proves the theorem. □

Degenerate Cases In some cases, the output polytope at fault-free nodes may be a single point, making the output equivalent to that obtained from vector consensus [54, 88]. As a trivial example, this occurs when all the fault-free nodes have identical input. It is possible to identify scenarios when the number of nodes is exactly equal to the lower bound, i.e., $n = (d+2)f + 1$ nodes, when the output polytope consists of just a single point. However, in general, particularly when n is larger than the lower bound, the output polytopes will contain infinite number of points. In any event, as shown in Theorem 25, our algorithm achieves optimality in all cases. Thus, any other algorithm can also produce such degenerate outputs for the same inputs.

9.6 Convex Hull Consensus under Crash Faults with Correct Inputs

With some simple changes, our algorithm and results can be extended to achieve convex hull consensus under the crash faults with correct inputs model. Under this model, we still need to satisfy the ϵ-agreement and termination properties stated in Chapter 9.1. The validity property remains unchanged as well; however, in this model, inputs at all the nodes are always correct. Thus, validity implies that the output will be contained in the convex hull of the inputs at all the nodes.

To obtain the algorithm for convex hull consensus under the crash faults with correct inputs model, two key changes required. First, the lower bound on the number of nodes becomes $n \geq 2f + 1$, which is independent of the dimension d. Second, instead of the computation in line 5 of Algorithm CC, the computation of $h_i[0]$ needs to be modified as $h_i[0] := \mathcal{H}(X_i)$, where $X_i := \{\, x \mid (x, k, 0) \in R_i\}$. With these changes, the modified algorithm achieves convex hull consensus under the crash faults with *correct* inputs model, with the rest of the proof being similar to the proof for the crash faults with *incorrect* inputs model. The modified algorithm exhibits optimal resilience as well.

9.7 Convex Hull Consensus under Byzantine Faults

As discussed in Chapter 9.1.1, the simulation techniques presented in [20, 5] can be used to transform our algorithm to an algorithm that tolerates Byzantine faults. This is because that our model assumes incorrect inputs at faulty nodes. In our technical report [77], we proposed an alternative algorithm, Optimal Verified Averaging, which uses *reliable broadcast* primitive [2] and *stable vector* primitive [4, 55], and also achieves optimal output polytope in the presence of Byzantine faults.

Optimal Verified Averaging proceeds in asynchronous rounds. In the initial round, we use stable vector primitive to exchange inputs. This ensures that enough fault-free nodes have observed the same (sub)set of the inputs – this is critical for proving optimality of the output polytope. In the subsequent rounds, we use a technique named verification to ensures that if a faulty node deviates from the algorithm specification (except possibly choosing an invalid input vector), then its incorrect messages will be ignored by the fault-free nodes. The verification mechanism is motivated by prior work by other researchers [20, 5]. With verification, aside from choosing a bad input, a faulty node cannot cause any other damage to the execution. The structure and the proof are similar to the ones of Algorithm CC and are thus presented in [77].

9.8 Convex Hull Function Optimization

The goal of *convex hull function optimization* is to minimize a cost function, say function c, over a domain consisting of the convex hull of the correct inputs. Formally, the following four properties must be satisfied by the function optimization algorithm:

- **Validity**: output y_i at fault-free node i is a point in the convex hull of the correct inputs.
- **ϵ-Agreement**: for any constant $\epsilon > 0$, for any fault-free nodes i, j, $\mathbf{d}_E(y_i, y_j) < \epsilon$.
- **Weak β-Optimality**: (i) for any constant $\beta > 0$, for any fault-free nodes i, j, $\|c(y_i) - c(y_j)\| < \beta$, and (ii) if at least $2f + 1$ nodes (faulty or fault-free) have an identical input, say x, then for any fault-free node i, $c(y_i) \leq c(x)$.
- **Termination:** each fault-free node must terminate within a finite amount of time.

The intuition behind part (ii) of the weak optimality condition above is as follows. When $2f + 1$ nodes have an identical input, say x^*, even if f of them are slow (or crash), each fault-free node must be able to learn that $f + 1$ nodes have input x^*, and at least one of these $f + 1$ nodes must be fault-free. Therefore, each fault-free node would know that the minimum value of the cost function over the convex hull of the correct inputs is at most $c(x^*)$.

It turns out that it is not feasible to simultaneously reach (approximate) consensus on a point, and to ensure that the cost function at that point is "small enough" for any arbitrary cost function.[2] The theorem below states this observation more formally.

Theorem 26 *The four properties of convex hull function optimization cannot be satisfied simultaneously in an asynchronous system in the presence of crash faults with incorrect inputs for $n \geq 4f + 1$ and $d \geq 1$.*

Proof: We will prove the result for $d = 1$. It should be obvious that impossibility with $d = 1$ implies impossibility for larger d (since we can always choose inputs that have 0 coordinates in all dimensions except one).

The proof is by contradiction. Suppose that there exists an algorithm, say Algorithm $\mathcal{A}$, that achieves the above four properties for $n \geq 4f + 1$ and $d = 1$.

Let the cost function be given by $c(x) = 4 - (2x - 1)^2$ for $x \in [0, 1]$ and $c(x) = 3$ for $x \notin [0, 1]$. For future reference, note that within the interval $[0, 1]$, function $c(x)$ has the smallest value at $x = 0, 1$ both.

Now, suppose that all the inputs (correct and incorrect) are restricted to be binary, and must be 0 or 1. We will prove impossibility under this restriction on the inputs at faulty and fault-free nodes both, which suffices to prove that the four properties cannot *always* be satisfied. Suppose that the output of Algorithm $\mathcal{A}$ at fault-free node i is y_i. Due to the validity property, and because the inputs are restricted to be 0 or 1, we know that $y_i \in [0, 1]$.

Since $\lceil \frac{n}{2} \rceil \geq \lceil \frac{4f+1}{2} \rceil = 2f + 1$, at least $2f + 1$ nodes will have either input 0, or input 1. Without loss of generality, suppose that at least $2f + 1$ nodes have input 0.

Consider a fault-free node i. By weak β-Optimality, $c(y_i) \leq c(0)$, that is, $c(y_i) \leq 3$. However, the minimum value of the cost function is 3 over all possible inputs. Thus, $c(y_i) = 3$. Similarly, for any other fault-free node j as well, $c(y_j)$ must equal 3. Now, due to validity, $y_j \in [0, 1]$, and the cost function is 3 in interval $[0, 1]$ only at $x = 0, 1$. Therefore, we must have y_i equal to 0 or 1, and y_j also equal to 0 or 1. However, because algorithm $\mathcal{A}$ satisfies the ϵ-agreement condition, $\mathbf{d}_E(y_i, y_j) = \|y_i - y_j\| < \epsilon$ (recall that dimension $d = 1$). If $\epsilon < 1$, then y_i and y_j must be identical (because we already know that they are either 0 or 1). Since this condition holds for any pair of fault-free nodes, it implies *exact* consensus. Also, y_i and y_j will be equal to the input at a fault-free node due to the validity property above, and because the inputs are restricted to be 0 or 1. In other words, Algorithm $\mathcal{A}$ can be used to solve exact consensus in the presence

[2] Impossibility result can be easily extended to the case when condition (ii) is relaxed as follows: $c(y_i) \leq c(x) + \beta'$ for some $\beta' > 0$. For brevity, we consider only the case when $c(y_i) \leq c(x)$ in this work.

of crash faults with incorrect inputs when $n \geq 4f+1$ in an asynchronous system. This contradicts the well-known impossibility result by Fischer, Lynch, and Paterson [32]. □

We know that even without the weak β-optimality, we need $n \geq (d+2)f+1$. Thus, the impossibility result is complete for $d \geq 2$. Whether the impossibility extends to $3f+1 \leq n \leq 4f$ and $d=1$ is presently unknown.

The natural question then is "What function optimization problem can we solve?" Suppose that the cost function satisfies b-Lipschitz continuity. That is, for any points x, y, $\|c(x) - c(y)\| \leq b\,\mathbf{d}_E(x,y)$. Below, we present an algorithm that achieves validity, weak β-optimality and termination, but not ϵ-agreement. The proposed algorithm has two simple steps:

- Step 1: First solve convex hull consensus with parameter ϵ. Let h_i be the output polytope of convex hull consensus at node i.
- Step 2: The output of function optimization is the tuple $(y_i, c(y_i))$, where $y_i = \arg\min_{x \in h_i} c(x)$. When there are multiple points in h_i minimizing $c(x)$, break tie arbitrarily.

The ϵ-agreement property of the convex hull consensus together with the assumption of b-Lipschitz continuity imply that for fault-free nodes i, j, $\|c(y_i) - c(y_j)\| < \epsilon b$. Thus, the fault-free nodes find approximately equal minimum value for the function. Therefore, for any $\beta > 0$, we can achieve $\|c(y_i) - c(y_j)\| < \beta$ by choosing $\epsilon = \beta/b$ for convex hull consensus in Step 1. Validity and termination follow directly form the properties of the convex hull consensus algorithm. Note that since in Step 2, nodes break tie arbitrarily, we are not able to guarantee that $\mathbf{d}_E(y_i, y_j)$ is small. That is, ϵ-agreement may not hold.

Notion of Optimality Observe that in the 2-step algorithm above, $c(y_i)$ at node i may not be minimum over the *entire* convex hull of the inputs of fault-free nodes. For instance, even when all the nodes are fault-free, each subset of f nodes is viewed as *possibly* faulty with incorrect inputs. We can extend the notion of optimality from Chapter 9.1 to function optimization as follows. An algorithm A for function optimization is said to be optimal if the following condition is true.

> Let F denote a set of up to f faulty nodes. For a *given execution* of algorithm A with F being the set of faulty nodes, let $y_i(A)$ denote the output at node i at the end of the given execution. For any other algorithm B, *there exists* an execution with F being the set of faulty nodes, such that $y_i(B)$ is the output at fault-free node i, and $c(y_j(A)) \leq c(y_j(B))$ for *each* fault-free node j.

The intuition behind the above formulation is as follows. A goal of function optimization here is to allow the nodes to "learn" the smallest value of the cost function over the convex hull of the inputs at the fault-free nodes. The above condition implies that an optimal algorithm will learn a function value that is no larger than that learned in a worst-case execution of any other algorithm.

The 2-step function optimization algorithm above is optimal in the above sense. This is a direct consequence of Theorem 25.

9.9 Summary

In this Chapter, we introduce the *convex hull consensus* problem under crash faults with incorrect inputs model, and present an asynchronous approximate convex hull consensus algorithm with optimal fault tolerance that reaches consensus on an optimal output polytope. We also consider the use of convex hull consensus algorithm to solve the problem of optimizing a function over the convex hull of the inputs at fault-free nodes. An impossibility result for asynchronous function optimization for arbitrary cost functions is also presented.

Chapter 10

Conclusions

This Chapter summarizes the dissertation and proposes future research directions.

10.1 Dissertation Summary

In this dissertation, we prove tight necessary and sufficient condition of the underlying communication networks for solving various kinds of consensus problems. We also discuss a new consensus problem – convex hull consensus. More specifically, in Chapter 3, we explore using general algorithms to achieve exact and approximate consensus in synchronous and asynchronous systems, respectively, and we consider f-total fault model with both crash-prone and Byzantine nodes. Then, we study using iterative algorithms to achieve approximate consensus in both synchronous and asynchronous systems. In particular, we explore the consensus problem under the following types of fault models:

- f-total fault model with Byzantine nodes in synchronous and asynchronous systems (Chapter 4)
- Generalized fault model with Byzantine nodes in synchronous systems (Chapter 5)
- Transient Byzantine link fault model in synchronous systems (Chapter 6)
- f-total fault model with crash-prone nodes in asynchronous systems (Chapter 7)

For this part of the work, we develope a proof technique based on famous matrix tools [38, 93, 9, 34] to prove the correctness of fault-tolerant iterative algorithms. The proof technique may be applied to relevant topics.

In Chapter 8, we consider using certified propagation algorithm to achieve *reliable broadcast*, in which a single fault-free source needs to transmit an input to all fault-free peers. For reliable broadcast, we assume f-local Byzantine fault model.

Finally, we present a new consensus problem – convex hull consensus – in which each node has a d-dimensional vector of reals as input ($d \geq 1$). Recall that the *initial convex hull* is defined as the convex hull of the inputs at fault-free nodes. *Convex hull consensus* requires that the output at each node is a *convex polytope* contained within the initial convex hull. Intuitively, the goal is to reach consensus on the "largest possible" polytope, allowing the node to estimate the domain of inputs at the fault-free nodes. In Chapter 9, we study the convex hull consensus problem under crash faults with incorrect inputs model in complete graphs. Particularly, we present an asynchronous approximate convex hull consensus algorithm with optimal fault tolerance that reaches consensus on largest possible output polytope.

10.2 Future Work

There are many open problems and potential research questions:

- As identified in Table 1.1, tight conditions of directed graphs for solving problems below are still open:
  - *Using iterative algorithms to solve exact Byzantine consensus in synchronous systems.*
  - *Using general algorithms to solve approximate Byzantine consensus in asynchronous systems.*
- As discussed in Chapter 3.6, tight condition for the following multi-valued exact Byzantine consensus is open: *Using general algorithms to solve multi-valued exact Byzantine consensus with stronger version of validity property in synchronous systems.* Here, the stronger version of validity is the same as the one for binary exact consensus – output of every fault-free node equals the input of a fault-free node.
- Given the fault-tolerance parameter f, how to identify whether it is possible to achieve fault-tolerant consensus in a graph under different fault models ?
- Tight conditions for achieving k-set consensus, vector consensus, convex hull consensus in directed graphs are still open.
- How to extend the results in Chapters 5 and 6 (consensus in generalized fault model and transient link fault model, respectively) to asynchronous systems?
- How to extend the results on directed graphs under f-total fault models (Chapters 3, 4, and 7) to dynamic networks?
- The Byzantine consensus algorithm presented in Chapter 3 (Algorithm BC) has exponential time complexity. How to improve the efficiency?
- Are there other suitable optimality conditions and applications of convex hull consensus?

Appendix A

Asynchronous Iterative Approximate Byzantine Consensus

In this Appendix, we present the complete analysis of IABC algorithms in asynchronous systems.

A.1 Algorithm Structure

By the definition of asynchronous systems, each node may proceed at different rate. Thus, Dolev et al. developed an algorithm based on "rounds" such that nodes update once in each round [29]. In particular, we consider the structure of *Async-IABC Algorithm* below, which has the same structure as the algorithm in [29]. This algorithm structure differs from the one for synchronous systems in Chapter 4 in two important ways: (i) the messages containing states are now tagged by the round index to which the states correspond, and (ii) each node i waits to receive only $|N_i^-| - f$ messages containing states from round $t-1$ before computing the new state in round t.

Due to the asynchronous nature of the system, different nodes may potentially perform their t-th round at very different real times. Thus, the main difference between iteration and round is as following:

- Iteration is defined as fixed amount of real-time units. Hence, every node will be in the same iteration at any given real time.
- Round is defined as the time that each node updates its value[1]. Hence, every node may be in totally different rounds at any given real time in asynchronous systems.

In Async-IABC algorithm, each node i maintains state v_i, with $v_i[t]$ denoting the state of node i at the end of its t-th round. Initial state of node i, $v_i[0]$, is equal to the initial input provided to node i. At the start of the t-th round ($t > 0$), the state of node i is $v_i[t-1]$. Now, we describe the steps that should be performed by each node $i \in \mathcal{V}$ in its t-th round.

[1] With a slight abuse of terminology, we will use "value" and "state" interchangeably in this report.

Async-IABC Algorithm

1. *Transmit step:* Transmit current state $v_i[t-1]$ on all outgoing edges. The message is tagged by index $t-1$.

2. *Receive step:* Wait until the first $|N_i^-| - f$ messages tagged by index $t-1$ are received on the incoming edges (breaking ties arbitrarily). Values received in these messages form vector $r_i[t]$ of size $|N_i^-| - f$.

3. *Update step:* Node i updates its state using a transition function Z_i, where Z_i is a part of the specification of the algorithm, and takes as input the vector $r_i[t]$ and state $v_i[t-1]$.

$$v_i[t] = Z_i \ (\ r_i[t] \, , \, v_i[t-1] \ ) \tag{A.1}$$

We now define $U[t]$ and $\mu[t]$, assuming that $\mathcal{F}$ is the set of Byzantine faulty nodes, with the nodes in $\mathcal{V} - \mathcal{F}$ being fault-free.[2]

- $U[t] = \max_{i \in \mathcal{V}-\mathcal{F}} \ v_i[t]$. $U[t]$ is the largest state among the fault-free nodes at the end of the t-th round. Since the initial state of each node is equal to its input, $U[0]$ is equal to the maximum value of the initial input at the fault-free nodes.

- $\mu[t] = \min_{i \in \mathcal{V}-\mathcal{F}} \ v_i[t]$. $\mu[t]$ is the smallest state among the fault-free nodes at the end of the t-th round. $\mu[0]$ is equal to the minimum value of the initial input at the fault-free nodes.

The following conditions must be satisfied by an Async-IABC algorithm in the presence of up to f Byzantine faulty nodes:

- *Validity:* $\forall t > 0, \quad \mu[t] \geq \mu[t-1] \quad$ and $\quad U[t] \leq U[t-1]$

- *Convergence:* $\lim_{t \to \infty} \ U[t] - \mu[t] = 0$

The objective in this Appendix is to identify the necessary and sufficient conditions for the existence of a *correct* Async-IABC algorithm (i.e., satisfying the above validity and convergence conditions) for a given $G(\mathcal{V}, \mathcal{E})$ in any asynchronous system.

A.2 Notations

There are many notations used and will be introduced later in this Appendix. Here is a quick reference:

[2] For sets X and Y, $X - Y$ contains elements that are in X but not in Y. That is, $X - Y = \{i \mid i \in X, \ i \notin Y\}$.

- N_i^+, N_i^-: set of outgoing neighbors and incoming neighbors of some node i, respectively.
- $U[t], \mu[t]$: maximum value and minimum value of all the fault-free nodes at the end of round t, respectively.
- Z_i: a function specifying how node i updates its new value (algorithm specification).
- $N_i^{@}[t]$: set of incoming neighbors from whom node i actually received values at round $t \geq 1$.
- $r_i[t]$: set of values sent by $N_i^{@}[t]$.
- $N_i^*[t]$: set of incoming neighbors from whom node i actually used the values to update at round $t \geq 1$.

Note that by definition of Async-IABC algorithms, we have the following relationships: $N_i^*[t] \subset N_i^{@}[t] \subset N_i^-$. Moreover, $N_i^*[t]$ and $N_i^{@}[t]$ may change over the rounds, and N_i^- is a constant. Lastly, $|N_i^{@}[t]| = |N_i^-| - 2f$ and $|N_i^*[t]| = |N_i^{@}[t]| - f$ for any round $t \geq 1$.

A.3 Necessary Condition

In asynchronous systems, for an Async-IABC algorithm satisfying the the *validity* and *convergence* conditions to exist, the underlying graph $G(\mathcal{V}, \mathcal{E})$ must satisfy a necessary condition proved in this section. Recall that we have defined relations $\overset{a}{\Rightarrow}$ and $\overset{a}{\not\Rightarrow}$ defined in Definition 14 in Chapter 4.6. We will use them to define the *tight* condition:

Condition Async: Consider graph $G(\mathcal{V}, \mathcal{E})$. Let sets F, L, C, R form a partition of $\mathcal{V}$, such that L and R are both non-empty and $|F| \leq f$, then either $C \cup R \overset{a}{\Rightarrow} L$, or $L \cup C \overset{a}{\Rightarrow} R$.

Now, we prove that *Condition Async* is necessary.

Theorem 27 *If an* Async-IABC Algorithm *satisfies* validity *and* convergence *conditions in graph $G(\mathcal{V}, \mathcal{E})$, then $G(\mathcal{V}, \mathcal{E})$ satisfies* Condition Async.

Proof: The proof is by contradiction. Let us assume that a correct Async-IABC consensus algorithm exists, and $C \cup R \overset{a}{\not\Rightarrow} L$ and $L \cup C \overset{a}{\not\Rightarrow} R$. Thus, for any $i \in L$, $|N_i^- \cap (C \cup R)| < 2f + 1$, and for any $j \in R$, $|N_j^- \cap (L \cup C)| < 2f + 1$,

Also assume that the nodes in F (if F is non-empty) are all faulty, and the remaining nodes, in sets L, R, C, are fault-free. Note that the fault-free nodes are not necessarily aware of the identity of the faulty nodes.

Consider the case when (i) each node in L has input m, (ii) each node in R has input M, such that $M > m$, and (iii) each node in C, if C is non-empty, has an input in the range $[m, M]$.

At the start of round 1, suppose that the faulty nodes in F (if non-empty) send $m^- < m$ to outgoing neighbors in L, send $M^+ > M$ to outgoing neighbors in R, and send some arbitrary value in $[m, M]$ to outgoing neighbors in C (if C is non-empty). This behavior is possible since nodes in F are faulty. Note that $m^- < m < M < M^+$. Each fault-free node $k \in \mathcal{V} - \mathcal{F}$, sends to nodes in N_k^+ value $v_k[0]$ in round 1.

Consider any node $i \in L$. Denote $N_i' = N_i^- \cap (C \cup R)$. Since $C \cup R \overset{a}{\not\Rightarrow} L$, $|N_i'| \leq 2f$. Consider the situation where the delay between certain $w = \min(f, |N_i'|)$ nodes in N_i' and node i is arbitrarily large compared to all the other traffic (including messages from incoming neighbors in F). Consequently, $r_i[1]$ includes $|N_i'| - w \leq f$ values from N_i', since w messages from N_i' are delayed and thus ignored by node i. Recall that $N_i^@[1]$ is the set of nodes whose round 1 values are received by node i in time (i.e., before i finishes step 2 in Async-IABC). By the argument above, $N_i^@[1] \cap N_i' \leq f$.

Node i receives m^- from the nodes in $F \cap N_i^@[1]$, values in $[m, M]$ from the nodes in $N_i' \cap N_i^@[1]$, and m from the nodes in $\{i\} \cup (L \cap N_i^@[1])$.

Consider four cases:

- $F \cap N_i^@[1]$ and $N_i' \cap N_i^@[1]$ are both empty: In this case, all the values that i receives are from nodes in $\{i\} \cup (L \cap N_i^@[1])$, and are identical to m. By validity condition, node i must set its new state, $v_i[1]$, to be m as well.

- $F \cap N_i^@[1]$ is empty and $N_i' \cap N_i^@[1]$ is non-empty: In this case, since $|N_i' \cap N_i^@[1]| \leq f$, from i's perspective, it is possible that all the nodes in $N_i^@[1] \cap N_i'$ are faulty, and the rest of the nodes are fault-free. In this situation, the values sent to node i by the fault-free nodes (which are all in $\{i\} \cup (L \cap N_i^@[1])$) are all m, and therefore, $v_i[1]$ must be set to m as per the validity condition.

- $F \cap N_i^@[1]$ is non-empty and $N_i' \cap N_i^@[1]$ is empty: In this case, since $|F \cap N_i^@[1]| \leq f$, it is possible that all the nodes in $F \cap N_i^@[1]$ are faulty, and the rest of the nodes are fault-free. In this situation, the values sent to node i by the fault-free nodes (which are all in $\{i\} \cup (L \cap N_i^@[1])$) are all m, and therefore, $v_i[1]$ must be set to m as per the validity condition.

- Both $F \cap N_i^@[1]$ and $N_i' \cap N_i^@[1]$ are non-empty: From node i's perspective, consider two possible scenarios: (a) nodes in $F \cap N_i^@[1]$ are faulty, and the other nodes are fault-free, and (b) nodes in $N_i' \cap N_i^@[1]$ are faulty, and the other nodes are fault-free.

  In scenario (a), from node i's perspective, the fault-free nodes have values in $[m, M]$ whereas the faulty nodes have value m^-. According to the validity condition, $v_i[1] \geq m$. On the other hand, in scenario (b), the fault-free nodes have values m^- and m, where $m^- < m$; so $v_i[1] \leq m$, according to the validity condition. Since node i does not know whether the correct scenario is (a) or (b), it must update its state to satisfy the validity condition in both cases. Thus, it follows that $v_i[1] = m$.

Observe that in each case above $v_i[1] = m$ for each node $i \in L$. Similarly, we can show that $v_j[1] = M$ for each node $j \in R$.

Now consider the nodes in set C, if C is non-empty. All the values received by the nodes in C are in $[m, M]$, therefore, their new state must also remain in $[m, M]$, as per the validity condition.

The above discussion implies that, at the end of the first round, the following conditions hold true: (i) state of each node in L is m, (ii) state of each node in R is M, and (iii) state of each node in C is in $[m, M]$. These conditions are identical to the initial conditions listed previously. Then, by induction, it follows that for any $t \geq 0$, $v_i[t] = m, \forall i \in L$, and $v_j[t] = M, \forall j \in R$. Since L and R contain fault-free nodes, the convergence requirement is not satisfied. This is a contradiction to the assumption that a correct Async-IABC algorithm exists. □

Corollary 6 *Let F, L, R be a partition of $\mathcal{V}$, such that $0 \leq |F| \leq f$, and L and R are non-empty. Then, either $L \overset{a}{\Rightarrow} R$ or $R \overset{a}{\Rightarrow} L$.*

Proof: The proof follows by setting $C = \emptyset$ in Theorem 27. □

Corollary 7 *The number of nodes n must exceed $5f$ for the existence of a correct Async-IABC algorithm that tolerates f Byzantine fault.*

Proof: The proof is by contradiction. Suppose that $2 \leq n \leq 5f$, and consider the following two cases:

- $2 \leq n \leq 4f$: Suppose that L, R, F is a partition of $\mathcal{V}$ such that $|L| = \lceil n/2 \rceil \leq 2f$, $|R| = \lfloor n/2 \rfloor \leq 2f$ and $F = \emptyset$. Note that L and R are non-empty, and $|L| + |R| = n$.

- $4f < n \leq 5f$:

  Suppose that L, R, F is a partition of $\mathcal{V}$, such that $|L| = |R| = 2f$ and $|F| = n - 4f$. Note that $0 < |F| \leq f$.

In both cases above, Corollary 6 is applicable. Thus, either $L \overset{a}{\Rightarrow} R$ or $R \overset{a}{\Rightarrow} L$. For $L \overset{a}{\Rightarrow} R$ to be true, L must contain at least $2f + 1$ nodes. Similarly, for $R \overset{a}{\Rightarrow} L$ to be true, R must contain at least $2f + 1$ nodes. Therefore, at least one of the sets L and R must contain more than $2f$ nodes. This contradicts our choice of L and R above (in both cases, size of L and R is $\leq 2f$). Therefore, n must be larger than $5f$. □

Corollary 8 *For the existence of a correct Async-IABC algorithm, then for each node $i \in \mathcal{V}$, $|N_i^-| \geq 3f+1$, i.e., each node i has at least $3f+1$ incoming links, when $f > 0$.*

Proof: The proof is by contradiction. Consider the following two cases for some node i:

- $|N_i^-| \leq 2f$: Define set $F = \emptyset, L = \{i\}$ and $R = V - F - L = V - \{i\}$. Thus, $N_i^- \cap R = N_i^-$, and $|N_i^- \cap R| \leq 2f$ by assumption.

- $2f < |N_i^-| \leq 3f$: Define set $L = \{i\}$. Partition N_i^- into two sets F and H such that $|F| = f$ and $|H| = |N_i^-| - f \leq 2f$. Define $R = V - F - L = V - F - \{i\}$. Thus, $N_i^- \cap R = H$, and $|N_i^- \cap R| \leq 2f$ by construction.

In both cases above, L and R are non-empty, so Corollary 6 is applicable. However, in each case, $L = \{i\}$ and $|L| = 1 < 2f+1$; hence, $L \overset{a}{\not\Rightarrow} R$. Also, since $L = \{i\}$ and $|N_i^- \cap R| \leq 2f$, and hence $R \overset{a}{\not\Rightarrow} L$ by the definition of $\overset{a}{\Rightarrow}$. This leads to a contradiction. Hence, every node must have at least $3f+1$ incoming neighbors. □

A.4 Useful Lemmas

In this section, we introduce two lemmas that are used in our proof of convergence. Note that the proofs are similar to corresponding lemmas in Chapter 4 except for the adoption of $\overset{a}{\Rightarrow}$ and "rounds" instead of $\Rightarrow$ and "iterations."

Definition 26 *For disjoint sets A, B, $in(A \overset{a}{\Rightarrow} B)$ denotes the set of all the nodes in B that each have at least $2f+1$ incoming links from nodes in A. More formally,*

$$in(A \overset{a}{\Rightarrow} B) = \{\ v\ |v \in B \text{ and }\ 2f+1 \leq |N_v^- \cap A|\ \}$$

With a slight abuse of notation, when $A \overset{a}{\not\Rightarrow} B$, define $in(A \overset{a}{\Rightarrow} B) = \emptyset$.

For brevity, we use the same name to define the "propagating sequences". The following definition should not be confused with the one defined in Definition 13.

Definition 27 *For* non-empty disjoint *sets A and B, set A is said to* **propagate to** *set B in l rounds, where $l > 0$, if there exist sequences of sets $A_0, A_1, A_2, \cdots, A_l$ and $B_0, B_1, B_2, \cdots, B_l$ (propagating sequences) such that*

- $A_0 = A$, $B_0 = B$, $B_l = \emptyset$, *and, for* $\tau < l$, $B_\tau \neq \emptyset$.

- *for* $0 \leq \tau \leq l-1$,

  * $A_\tau \overset{a}{\Rightarrow} B_\tau$,

  * $A_{\tau+1} = A_\tau \cup in(A_\tau \overset{a}{\Rightarrow} B_\tau)$, *and*

  * $B_{\tau+1} = B_\tau - in(A_\tau \overset{a}{\Rightarrow} B_\tau)$

Observe that A_τ and B_τ form a partition of $A \cup B$, and for $\tau < l$, $in(A_\tau \overset{a}{\Rightarrow} B_\tau) \neq \emptyset$. Also, when set A propagates to set B, length l above is necessarily finite. In particular, l is upper bounded by $n - 2f - 1$, since set A must be of size at least $2f + 1$ for it to propagate to B.

Note that in the proof of the following useful lemmas, there is no notion of iteration involves. In other words, the message delay does not affect the correctness of the proof. As long as message can be delivered in order and in finite amount of time, the proof follows. Thus, using the definitions, we have the following two lemmas for asynchronous algorithm.

There is one subtle concept worthy of some discussion. The propagating sequence is a global view of the system[3]. In synchronous system in Chapter 4, there is not much confusion, since global time is consistent with local time at each node. In asynchronous system, the propagating sequence is still a global view, but it is a view with respect to the notion of "round" instead of real time (measured by some external clock). For example, τ here means all the node values in round τ, and should not be confused with some real time τ.

Lemma 59 *Assume that* $G(\mathcal{V}, \mathcal{E})$ *satisfies* Condition Async. *Consider a partition* A, B, F *of* $\mathcal{V}$ *such that* A *and* B *are non-empty, and* $|F| \leq f$. *If* $B \overset{a}{\not\Rightarrow} A$, *then set* A *propagates to set* B.

Proof: Since A, B are non-empty, and $B \overset{a}{\not\Rightarrow} A$, by Corollary 6, we have $A \overset{a}{\Rightarrow} B$.

The proof is by induction. Define $A_0 = A$ and $B_0 = B$. Thus $A_0 \overset{a}{\Rightarrow} B_0$ and $B_0 \overset{a}{\not\Rightarrow} A_0$. Note that A_0 and B_0 are non-empty.

Induction basis: For some $\tau \geq 0$,

- for $0 \leq k < \tau$, $A_k \overset{a}{\Rightarrow} B_k$, and $B_k \neq \emptyset$,

- either $B_\tau = \emptyset$ or $A_\tau \overset{a}{\Rightarrow} B_\tau$,

- for $0 \leq k < \tau$, $A_{k+1} = A_k \cup in(A_k \overset{a}{\Rightarrow} B_k)$, and $B_{k+1} = B_k - in(A_k \overset{a}{\Rightarrow} B_k)$

Since $A_0 \overset{a}{\Rightarrow} B_0$, the induction basis holds true for $\tau = 0$.

[3] Such concept is just for the analysis, and each node does not need to know the global view.

Induction: If $B_\tau = \emptyset$, then the proof is complete, since all the conditions specified in Definition 27 are satisfied by the sequences of sets $A_0, A_1, \cdots, A_\tau$ and $B_0, B_1, \cdots, B_\tau$.

Now consider the case when $B_\tau \neq \emptyset$. By assumption, $A_k \overset{a}{\Rightarrow} B_k$, for $0 \leq k \leq \tau$. Define $A_{\tau+1} = A_\tau \cup in(A_\tau \overset{a}{\Rightarrow} B_\tau)$ and $B_{\tau+1} = B_\tau - in(A_\tau \overset{a}{\Rightarrow} B_\tau)$. Our goal is to prove that either $B_{\tau+1} = \emptyset$ or $A_{\tau+1} \overset{a}{\Rightarrow} B_{\tau+1}$. If $B_{\tau+1} = \emptyset$, then the induction is complete. Therefore, now let us assume that $B_{\tau+1} \neq \emptyset$ and prove that $A_{\tau+1} \overset{a}{\Rightarrow} B_{\tau+1}$. We will prove this by contradiction.

Suppose that $A_{\tau+1} \overset{a}{\not\Rightarrow} B_{\tau+1}$. Define subsets L, C, R as follows: $L = A_0$, $C = A_{\tau+1} - A_0$ and $R = B_{\tau+1}$. Due to the manner in which A_k's and B_k's are defined, we also have $C = B_0 - B_{\tau+1}$. Observe that L, C, R, F form a partition of $\mathcal{V}$, where L, R are non-empty, and the following relationships hold:

- $C \cup R = B_0$, and
- $L \cup C = A_{\tau+1}$

Rewriting $B_0 \overset{a}{\not\Rightarrow} A_0$ and $A_{\tau+1} \overset{a}{\not\Rightarrow} B_{\tau+1}$, using the above relationships, we have, respectively,

$$C \cup R \overset{a}{\not\Rightarrow} L,$$

and

$$L \cup C \overset{a}{\not\Rightarrow} R$$

This violates *Condition Async*. This is a contradiction, completing the induction.

Thus, we have proved that, either (i) $B_{\tau+1} = \emptyset$, or (ii) $A_{\tau+1} \overset{a}{\Rightarrow} B_{\tau+1}$. Eventually, for large enough t, B_t will become $\emptyset$, resulting in the propagating sequences $A_0, A_1, \cdots, A_t$ and $B_0, B_1, \cdots, B_t$, satisfying the conditions in Definition 27. Therefore, A propagates to B. □

Lemma 60 *Assume that $G(\mathcal{V}, \mathcal{E})$ satisfies* Condition Async*. For any partition A, B, F of $\mathcal{V}$, where A, B are both non-empty, and $|F| \leq f$, at least one of the following conditions must be true:*

- *A propagates to B, or*
- *B propagates to A*

Proof: Consider two cases:

- $A \overset{a}{\not\Rightarrow} B$: Then by Lemma 59, B propagates to A, completing the proof.
- $A \overset{a}{\Rightarrow} B$: In this case, consider two sub-cases:

- A propagates to B: The proof in this case is complete.
- A does not propagate to B: Thus, propagating sequences defined in Definition 27 do not exist in this case. More precisely, there must exist $k > 0$, and sets $A_0, A_1, \cdots, A_k$ and $B_0, B_1, \cdots, B_k$, such that:
  - $A_0 = A$ and $B_0 = B$, and
  - for $0 \leq i \leq k-1$,
    - $A_i \overset{a}{\Rightarrow} B_i$,
    - $A_{i+1} = A_i \cup in(A_i \overset{a}{\Rightarrow} B_i)$, and
    - $B_{i+1} = B_i - in(A_i \overset{a}{\Rightarrow} B_i)$.
  - $B_k \neq \emptyset$ and $A_k \overset{a}{\not\Rightarrow} B_k$.

  The last condition above violates the requirements for A to propagate to B.

  Now $A_k \neq \emptyset$, $B_k \neq \emptyset$, and A_k, B_k, F form a partition of $\mathcal{V}$. Since $A_k \overset{a}{\not\Rightarrow} B_k$, by Lemma 59, B_k propagates to A_k.

  Since $B_k \subseteq B_0 = B$, $A \subseteq A_k$, and B_k propagates to A_k, it should be easy to see that B propagates to A.

□

A.5 Sufficient Condition

A.5.1 Algorithm 5

We will prove that there exists an Async-IABC algorithm – particularly *Algorithm 5* below – that satisfies the *validity* and *convergence* conditions provided that the graph $G(\mathcal{V}, \mathcal{E})$ satisfies *Condition Async.* This implies that *Condition Async* is also sufficient.

Algorithm 5

1. *Transmit step:* Transmit current state $v_i[t-1]$ on all outgoing edges.
2. Receive step: Wait until receiving values on all but f incoming edges. These values form vector $r_i[t]$ of size $|N_i^-| - f$.[4]
3. *Update step:* Sort the values in $r_i[t]$ in an increasing order, and eliminate the smallest f values, and the largest f values (breaking ties arbitrarily). Let $N_i^*[t]$ denote the identifiers of nodes from whom

[4] If more than $|N_i^-| - f$ values arrive at the same time, break ties arbitrarily.

the remaining $N_i^- - 3f$ values were received, and let w_j denote the value received from node $j \in N_i^*$. For convenience, define $w_i = v_i[t-1]$ to be the value node i "receives" from itself. Observe that if $j \in \{i\} \cup N_i^*[t]$ is fault-free, then $w_j = v_j[t-1]$.

Define

$$v_i[t] \; = \; Z_i(r_i[t], v_i[t-1]) \; = \sum_{j \in \{i\} \cup N_i^*[t]} a_i \, w_j \tag{A.2}$$

where

$$a_i = \frac{1}{|N_i^-| + 1 - 3f}$$

Note that $|N_i^*[t]| = |N_i^-| - 3f$, and $i \notin N_i^*[t]$ because $(i,i) \notin \mathcal{E}$. The "weight" of each term on the right-hand side of (A.2) is a_i, and these weights add to 1. Also, $0 < a_i \leq 1$. For future reference, let us define α as:

$$\alpha = \min_{i \in \mathcal{V}} \; a_i \tag{A.3}$$

A.5.2 Sufficiency

In Theorems 28 and 29 in this section, we prove that Algorithm 5 satisfies *validity* and *convergence* conditions, respectively, provided that $G(\mathcal{V}, \mathcal{E})$ satisfies *Condition Async.*

Note that the proofs below are similar to the ones presented in Chapter 4.5. The main differences are the following:

- We need to consider only values in $N_i^@[t]$ not in N_i^-. This is due to different step 2 between Algorithm 1 (Chapter 4.4) and Algorithm 5.
- We interpret t as round index, rather than iteration index.

Theorem 28 *Suppose that $G(\mathcal{V}, \mathcal{E})$ satisfies* Condition Async*. Then Algorithm 5 satisfies the* validity *condition.*

Proof: Consider the t-th round, and any fault-free node $i \in \mathcal{V} - \mathcal{F}$. Consider two cases:

- $f = 0$: In (A.2), note that $v_i[t]$ is computed using states from the previous round at node i and other nodes. By definition of $\mu[t-1]$ and $U[t-1]$, $v_j[t-1] \in [\mu[t-1], U[t-1]]$ for all fault-free nodes $j \in \mathcal{V} - \mathcal{F}$. Thus, in this case, all the values used in computing $v_i[t]$ are in the range $[\mu[t-1], U[t-1]]$. Since $v_i[t]$ is computed as a weighted average of these values, $v_i[t]$ is also within $[\mu[t-1], U[t-1]]$.

- $f > 0$: By Corollary 8, $|N_i^-| \geq 3f+1$. Thus, $|N_i^@| \geq 2f+1$, and $|r_i[t]| \geq 2f+1$. When computing set $N_i^*[t]$, the largest f and smallest f values from $r_i[t]$ are eliminated. Since at most f nodes are faulty, it follows that, either (i) the values received from the faulty nodes are all eliminated, or (ii) the values from the faulty nodes that still remain are between values received from two fault-free nodes. Thus, the remaining values in $r_i[t]$ are all in the range $[\mu[t-1], U[t-1]]$. Also, $v_i[t-1]$ is in $[\mu[t-1], U[t-1]]$, as per the definition of $\mu[t-1]$ and $U[t-1]$. Thus $v_i[t]$ is computed as a weighted average of values in $[\mu[t-1], U[t-1]]$, and, therefore, it will also be in $[\mu[t-1], U[t-1]]$.

Since $\forall i \in \mathcal{V} - \mathcal{F}$, $v_i[t] \in [\mu[t-1], U[t-1]]$, the validity condition is satisfied. □

Before proving the convergence of Algorithm 5, we first present three lemmas. In the discussion below, we assume that $G(\mathcal{V}, \mathcal{E})$ satisfies the sufficient condition.

Lemma 61 *Consider node $i \in \mathcal{V} - \mathcal{F}$. Let $\psi \leq \mu[t-1]$. Then, for $j \in \{i\} \cup N_i^*[t]$,*

$$v_i[t] - \psi \geq a_i \ (w_j - \psi)$$

Specifically, for fault-free $j \in \{i\} \cup N_i^*[t]$,

$$v_i[t] - \psi \geq a_i \ (v_j[t-1] - \psi)$$

Proof: In (A.2), for each $j \in N_i^*[t]$, consider two cases:

- Either $j = i$ or $j \in N_i^*[t] \cap (\mathcal{V} - \mathcal{F})$: Thus, j is fault-free. In this case, $w_j = v_j[t-1]$. Therefore, $\mu[t-1] \leq w_j \leq U[t-1]$.

- j is faulty: In this case, f must be non-zero (otherwise, all nodes are fault-free). From Corollary 8, $|N_i^-| \geq 3f+1$. Thus, $|N_i^@| \geq 2f+1$, and $|r_i[t]| \geq 2f+1$. Then it follows that the smallest f values in $r_i[t]$ that are eliminated in step 2 of Algorithm 5 contain the state of at least one fault-free node, say k. This implies that $v_k[t-1] \leq w_j$. This, in turn, implies that $\mu[t-1] \leq w_j$.

Thus, for all $j \in \{i\} \cup N_i^*[t]$, we have $\mu[t-1] \leq w_j$. Therefore,

$$w_j - \psi \geq 0 \text{ for all } j \in \{i\} \cup N_i^*[t] \tag{A.4}$$

Since weights in Equation (A.2) add to 1, we can re-write that equation as,

$$\begin{aligned} v_i[t] - \psi &= \sum_{j \in \{i\} \cup N_i^*[t]} a_i \, (w_j - \psi) \\ &\geq a_i \, (w_j - \psi), \quad \forall j \in \{i\} \cup N_i^*[t] \qquad \text{from (A.4)} \end{aligned} \tag{A.5}$$

For fault-free $j \in \{i\} \cup N_i^*[t]$, $w_j = v_j[t-1]$, therefore,

$$v_i[t] - \psi \quad \geq \quad a_i \, (v_j[t-1] - \psi) \tag{A.6}$$

□

Similar to the above result, we can also show the following lemma:

Lemma 62 *Consider node $i \in \mathcal{V} - \mathcal{F}$. Let $\Psi \geq U[t-1]$. Then, for $j \in \{i\} \cup N_i^*[t]$,*

$$\Psi - v_i[t] \geq a_i \ (\Psi - w_j)$$

Specifically, for fault-free $j \in \{i\} \cup N_i^*[t]$,

$$\Psi - v_i[t] \geq a_i \ (\Psi - v_j[t-1])$$

Then we present the main lemma used in proof of convergence. Note that below, we use parameter α defined in (A.3). Recall that in (A.2) in Algorithm 5, $a_i > 0$ for all i, and thus, $\alpha > 0$.

Lemma 63 *At the end of the s-th round, suppose that the fault-free nodes in $\mathcal{V} - \mathcal{F}$ can be partitioned into non-empty sets R and L such that (i) R propagates to L in l rounds, and (ii) the states of nodes in R are confined to an interval of length $\leq \frac{U[s]-\mu[s]}{2}$. Then,*

$$U[s+l] - \mu[s+l] \leq \left(1 - \frac{\alpha^l}{2}\right) (U[s] - \mu[s]) \tag{A.7}$$

Proof: Since R propagates to L, as per Definition 27, there exist sequences of sets $R_0, R_1, \cdots, R_l$ and $L_0, L_1, \cdots, L_l$, where

- $R_0 = R$, $L_0 = L$, $L_l = \emptyset$, for $0 \leq \tau < l$, $L_\tau \neq \emptyset$, and

- for $0 \leq \tau \leq l-1$,
    - * $R_\tau \overset{a}{\Rightarrow} L_\tau$,
    - * $R_{\tau+1} = R_\tau \cup in(R_\tau \overset{a}{\Rightarrow} L_\tau)$, and
    - * $L_{\tau+1} = L_\tau - in(R_\tau \overset{a}{\Rightarrow} L_\tau)$

Let us define the following bounds on the states of the nodes in R at the end of the s-th round:

$$\begin{aligned} M &= max_{j \in R}\ v_j[s] && \text{(A.8)} \\ m &= min_{j \in R}\ v_j[s] && \text{(A.9)} \end{aligned}$$

By the assumption in the statement of Lemma 63,

$$M - m \leq \frac{U[s] - \mu[s]}{2} \tag{A.10}$$

Also, $M \leq U[s]$ and $m \geq \mu[s]$. Therefore, $U[s] - M \geq 0$ and $m - \mu[s] \geq 0$.

The remaining proof of Lemma 63 relies on derivation of the three intermediate claims below.

Claim 12 *For $0 \leq \tau \leq l$, for each node $i \in R_\tau$,*

$$v_i[s+\tau] - \mu[s] \geq \alpha^\tau (m - \mu[s]) \tag{A.11}$$

Proof of Claim 12: The proof is by induction.

Induction basis: For some τ, $0 \leq \tau < l$, for each node $i \in R_\tau$, (A.11) holds. By definition of m, the induction basis holds true for $\tau = 0$.

Induction: Assume that the induction basis holds true for some τ, $0 \leq \tau < l$. Consider $R_{\tau+1}$. Observe that R_τ and $R_{\tau+1} - R_\tau$ form a partition of $R_{\tau+1}$; let us consider each of these sets separately.

- Set R_τ: By assumption, for each $i \in R_\tau$, (A.11) holds true. By validity of Algorithm 5 (Theorem 28), $\mu[s] \leq \mu[s+\tau]$. Therefore, setting $\psi = \mu[s]$ in Lemma 61, we get,

$$\begin{aligned} v_i[s+\tau+1] - \mu[s] &\geq a_i\ (v_i[s+\tau] - \mu[s]) \\ &\geq a_i\ \alpha^\tau (m - \mu[s]) \qquad \text{due to (A.11)} \\ &\geq \alpha^{\tau+1} (m - \mu[s]) \qquad \text{due to (A.3)} \end{aligned}$$

- Set $R_{\tau+1} - R_\tau$: Consider a node $i \in R_{\tau+1} - R_\tau$. By definition of $R_{\tau+1}$, we have that $i \in in(R_\tau \overset{a}{\Rightarrow} L_\tau)$. Thus,

  $$|N_i^- \cap R_\tau| \geq 2f + 1$$

  It follows that

  $$|N_i^@[s+\tau] \cap R_\tau| \geq f + 1$$

  In Algorithm 5, $2f$ values (f smallest and f largest) received by node i are eliminated before $v_i[s+\tau+1]$ is computed at the end of $(s+\tau+1)$-th round. Consider two possibilities:

  – Value received from one of the nodes in $N_i^@[s+\tau] \cap R_\tau$ is **not** eliminated. Suppose that this value is received from fault-free node $p \in N_i^@[s+\tau] \cap R_\tau$. Then, by an argument similar to the previous case, we can set $\psi = \mu[s]$ in Lemma 61, to obtain,

  $$\begin{aligned} v_i[s+\tau+1] - \mu[s] &\geq a_i \ (v_p[s+\tau] - \mu[s]) \\ &\geq a_i \ \alpha^\tau (m - \mu[s]) \qquad \text{due to (A.11)} \\ &\geq \alpha^{\tau+1}(m - \mu[s]) \qquad \text{due to (A.3)} \end{aligned}$$

  – Values received from **all** (there are at least $f+1$) nodes in $N_i^@[s+\tau] \cap R_\tau$ are eliminated. Note that in this case f must be non-zero (for $f = 0$, no value is eliminated, as already considered in the previous case). By Corollary 8, we know that each node must have at least $3f+1$ incoming edges. Thus, $N_i^@[t+\tau] \geq 2f+1$. Since at least $f+1$ values from nodes in $N_i^@[t+\tau] \cap R_\tau$ are eliminated, and there are at least $2f+1$ values to choose from, it follows that the values that are **not** eliminated are within the interval to which the values from $N_i^@[s+\tau] \cap R_\tau$ belong. Thus, there exists a node k (possibly faulty) from whom node i receives some value w_k – which is not eliminated – and a fault-free node $p \in N_i^@[t+\tau] \cap R_\tau$ such that

  $$v_p[s+\tau] \quad \leq \quad w_k \tag{A.12}$$

Then by setting $\psi = \mu[s]$ in Lemma 61 we have

$$\begin{aligned} v_i[s+\tau+1] - \mu[s] &\geq a_i \ (w_k - \mu[s]) \\ &\geq a_i \ (v_p[s+\tau] - \mu[s]) \qquad \text{due to (A.12)} \\ &\geq a_i \ \alpha^{\tau}(m - \mu[s]) \qquad \text{due to (A.11)} \\ &\geq \alpha^{\tau+1}(m - \mu[s]) \qquad \text{due to (A.3)} \end{aligned}$$

Thus, we have shown that for all nodes in $R_{\tau+1}$,

$$v_i[s+\tau+1] - \mu[s] \geq \alpha^{\tau+1}(m - \mu[s])$$

This completes the proof of Claim 12.

Claim 13 *For each node $i \in \mathcal{V} - \mathcal{F}$,*

$$v_i[s+l] - \mu[s] \geq \alpha^{l}(m - \mu[s]) \tag{A.13}$$

Proof of Claim 12:

Note that by definition, $R_l = \mathcal{V} - \mathcal{F}$. Then the proof follows by setting $\tau = l$ in the above Claim 12.

By a procedure similar to the derivation of Claim 13 above, we can also prove the claim below.

Claim 14 *For each node $i \in \mathcal{V} - \mathcal{F}$,*

$$U[s] - v_i[s+l] \geq \alpha^{l}(U[s] - M) \tag{A.14}$$

Now let us resume the proof of the Lemma 63. Note that $R_l = \mathcal{V} - \mathcal{F}$. Thus,

$$\begin{aligned} U[s+l] &= \max_{i \in \mathcal{V} - \mathcal{F}} v_i[s+l] \\ &\leq U[s] - \alpha^{l}(U[s] - M) \qquad \text{by (A.14)} \end{aligned} \tag{A.15}$$

and

$$\begin{aligned}\mu[s+l] &= \min_{i\in\mathcal{V}-\mathcal{F}} v_i[s+l] \\ &\geq \mu[s]+\alpha^l(m-\mu[s]) \qquad \text{by (A.13)} \qquad (A.16)\end{aligned}$$

Subtracting (A.16) from (A.15),

$$\begin{aligned}U[s+l]-\mu[s+l] &\leq U[s]-\alpha^l(U[s]-M)-\mu[s]-\alpha^l(m-\mu[s]) \\ &= (1-\alpha^l)(U[s]-\mu[s])+\alpha^l(M-m) \qquad (A.17) \\ &\leq (1-\alpha^l)(U[s]-\mu[s])+\alpha^l\ \frac{U[s]-\mu[s]}{2} \qquad \text{by (A.10)} \qquad (A.18) \\ &\leq (1-\frac{\alpha^l}{2})(U[s]-\mu[s]) \qquad (A.19)\end{aligned}$$

This concludes the proof of Lemma 63.

□

Now, we are able to prove the convergence of Algorithm 5. Note that this proof is essentially identical to the synchronous case presented in Chapter 4. We include it here for completeness.

Theorem 29 *Suppose that $G(\mathcal{V},\mathcal{E})$ satisfies Condition Async. Then Algorithm 5 satisfies the* convergence *condition.*

Proof:

Our goal is to prove that, given any $\epsilon > 0$, there exists τ such that

$$U[t]-\mu[t] \leq \epsilon \quad \forall t \geq \tau \qquad (A.20)$$

Consider the s-th round, for some $s \geq 0$. If $U[s]-\mu[s]=0$, then the algorithm has already converged, and the proof is complete, with $\tau = s$.

Now consider the case when $U[s]-\mu[s]>0$. Partition $\mathcal{V}-\mathcal{F}$ into two subsets, A and B, such that, for each node $i \in A$, $v_i[s] \in \left[\mu[s], \frac{U[s]+\mu[s]}{2}\right)$, and for each node $j \in B$, $v_j[s] \in \left[\frac{U[s]+\mu[s]}{2}, U[s]\right]$. By definition of $\mu[s]$ and $U[s]$, there exist fault-free nodes i and j such that $v_i[s]=\mu[s]$ and $v_j[s]=U[s]$. Thus, sets A and B are both non-empty. By Lemma 60, one of the following two conditions must be true:

- Set A propagates to set B. Then, define $L=B$ and $R=A$. The states of all the nodes in $R=A$ are confined within an interval of length $< \frac{U[s]+\mu[s]}{2}-\mu[s] \leq \frac{U[s]-\mu[s]}{2}$.

- Set B propagates to set A. Then, define $L = A$ and $R = B$. In this case, states of all the nodes in $R = B$ are confined within an interval of length $\leq U[s] - \frac{U[s]+\mu[s]}{2} \leq \frac{U[s]-\mu[s]}{2}$.

In both cases above, we have found non-empty sets L and R such that (i) L, R is a partition of $\mathcal{V} - \mathcal{F}$, (ii) R propagates to L, and (iii) the states in R are confined to an interval of length $\leq \frac{U[s]-\mu[s]}{2}$. Suppose that R propagates to L in $l(s)$ steps, where $l(s) \geq 1$. By Lemma 63,

$$U[s + l(s)] - \mu[s + l(s)] \leq \left(1 - \frac{\alpha^{l(s)}}{2}\right)(U[s] - \mu[s]) \tag{A.21}$$

Since $n - f - 1 \geq l(s) \geq 1$ and $0 < \alpha \leq 1$, $0 \leq \left(1 - \frac{\alpha^{l(s)}}{2}\right) < 1$.

Let us define the following sequence of round indices[5]:

- $\tau_0 = 0$,
- for $i > 0$, $\tau_i = \tau_{i-1} + l(\tau_{i-1})$, where $l(s)$ for any given s was defined above.

By repeated application of the argument leading to (A.21), we can prove that, for $i \geq 0$,

$$U[\tau_i] - \mu[\tau_i] \leq \left(\Pi_{j=1}^{i}\left(1 - \frac{\alpha^{\tau_j - \tau_{j-1}}}{2}\right)\right)(U[0] - \mu[0]) \tag{A.22}$$

For a given ϵ, by choosing a large enough i, we can obtain

$$\left(\Pi_{j=1}^{i}\left(1 - \frac{\alpha^{\tau_j - \tau_{j-1}}}{2}\right)\right)(U[0] - \mu[0]) \leq \epsilon$$

and, therefore,

$$U[\tau_i] - \mu[\tau_i] \leq \epsilon \tag{A.23}$$

For $t \geq \tau_i$, by validity of Algorithm 5, it follows that

$$U[t] - \mu[t] \leq U[\tau_i] - \mu[\tau_i] \leq \epsilon$$

This concludes the proof. □

[5] Without loss of generality, we assume that $U[\tau_i] - \mu[\tau_i] > 0$. Otherwise, the statement is trivially true due to the validity shown in Theorem 28.

Appendix B

Asynchronous Iterative Approximate Crash-tolerant Consensus

B.1 Proof of Lemma 47

Proof: We first prove that *Condition ICCA* implies *Condition ICCA2*.

By assumption, G contains at least two node, and so does G_f; therefore, at least one source component must exist in G_f. We now prove that G_f cannot contain more than one source component. The proof is by contradiction. Suppose that there exists a reduced graph $G_f(\mathcal{V}, \mathcal{E}_f)$ such that the decomposition of G_f includes at least two source components.

Let the sets of nodes in two such source components of G_f be denoted L and R, respectively. Let $C = \mathcal{V} - L - R$. Observe that L, C, R form a partition of the nodes in $\mathcal{V}$. Since L is a source component in G_f, it follows that there are no directed links in $\mathcal{E}_f$ from any node in $C \cup R$ to the nodes in L. Similarly, since R is a source component in G_f, it follows that there are no directed links in $\mathcal{E}_f$ from any node in $L \cup C$ to the nodes in R. These observations, together with the manner in which $\mathcal{E}_f$ is defined, imply that (i) there are at most f links in $\mathcal{E}$ from the nodes in $C \cup R$ to each node in L, and (ii) there are at most f links in $\mathcal{E}$ from the nodes in $L \cup C$ to each node in R. Therefore, in graph $G' = (\mathcal{V}, \mathcal{E}_f)$, $C \cup R \not\Rightarrow L$ and $L \cup C \not\Rightarrow R$. Thus, $G = (\mathcal{V}, \mathcal{E})$ does not satisfies *Condition ICCA*, a contradiction.

Now, we prove that *Condition ICCA2* implies *Condition ICCA*.

The proof is by contradiction. Suppose that *Condition ICCA* does not hold for graph $G = (\mathcal{V}, \mathcal{E})$. Thus, there exist a node partition L, C, R, where L and R are both non-empty, such that $C \cup R \not\Rightarrow L$ and $L \cup C \not\Rightarrow R$.

We now constructed a reduced graph $G_F(\mathcal{V}, \mathcal{E}_f)$. Observe that since $C \cup R \not\Rightarrow L$, the number of links at each node in L from nodes in $C \cup R$ is at most f; remove all these links. Similarly, for every node $j \in R$, remove all links from nodes in $L \cup C$ to j (recall that by assumption, there are at most f such links). The remaining links form the set $\mathcal{E}_f$. It should be obvious that $G_F(\mathcal{V}, \mathcal{E}_f)$ satisfies Definition 21; hence, G_f is a valid reduced graph.

Now, observe that by construction, in the reduced graph $G_f(\mathcal{V}, \mathcal{E}_f)$, there are no incoming links to nodes in R from nodes in $L \cup C$; similarly, in G_f, there are no incoming links to nodes in L from nodes in $C \cup R$. It follows that for each $i \in L$, there is no path using links in $\mathcal{E}_f$ from i to nodes in R; similarly, for each $j \in R$, there is no path using links in $\mathcal{E}_f$ from j to nodes in L. Thus, G_f must contain at least two source components. Therefore, the existence of G_f implies that G violates *Condition ICCA2*, a contradiction. □

B.2 Correctness of Algorithm 4 (Theorem 18)

We prove that Algorithm 4 is correct and thus, show that Conditions IAAC and IAAC2 are sufficient.

Validity Property Consider equation (7.7). Since each $\mathbf{M}[u]$ is row stochastic as shown in Lemma 50, the matrix product $\Pi_{u=1}^{t}\mathbf{M}[u]$ is also a row stochastic matrix. Thus, (7.7) implies that the state of each node i at the end of iteration t can be expressed as a convex combination of the initial states at all the nodes. Therefore, the validity property is satisfied.

Termination Property Algorithm 4 terminates after t_{end} iterations, where t_{end} is a finite constant depending only on $G = (\mathcal{V}, \mathcal{E}), U, \mu$, and ϵ. Recall that U and μ are defined as upper and lower bounds of the initial inputs at all nodes, respectively. Therefore, trivially, the algorithm satisfies the termination property. Later, using (B.4), we define a suitable value for t_{end}.

ϵ-agreement Property Denote by R_f the set of all the reduced graph of $G(\mathcal{V}, \mathcal{E})$ corresponding to the bound on the number of faulty nodes f as per Definition 21. Let

$$r = |R_f|$$

Note that r only depends on $G(\mathcal{V}, \mathcal{E})$ and f, and is a finite integer.

Consider iteration t ($t \geq 1$). Then for each reduced graph $H[t] \in R_f$, define connectivity matrix $\mathbf{H}[t]$ as follows, where $1 \leq i, j \leq n$:

- $\mathbf{H}_{ij}[t] = 1$, if either $j = i$, or edge (j, i) exists in reduced graph H;
- $\mathbf{H}_{ij}[t] = 0$, otherwise.

Thus, the non-zero elements of row $\mathbf{H}_i[t]$ correspond to the incoming links at node i in the reduced graph $H[t]$, or the self-loop at i. Observe that $\mathbf{H}[t]$ has a non-zero diagonal.

Based on *Condition ICCA2* and Lemmas 48, 50, we can show the following key lemmas.

Lemma 64 *For any $H[t] \in R_f$, and $k \geq n$, $\mathbf{H}^k[t]$ has at least one non-zero column, i.e., a column with all elements non-zero.*

Proof: $G(\mathcal{V}, \mathcal{E})$ satisfies the *Condition ICCA2*. Therefore, by Lemma 48, there exists at least one node p in the reduced graph $H[t]$ that has directed paths to all the nodes in $H[t]$ (consisting of the edges in $H[t]$). $\mathbf{H}^k_{jp}[t]$ of product $\mathbf{H}^k[t]$ is 1 if and only if node p has a directed path to node j consisting of at most k edges in $H[t]$. Since the length of the path from p to any other node in $H[t]$ is at most n, and p has directed paths to all the nodes, for $k \geq n$ the p-th column of matrix $\mathbf{H}^k[t]$ will be non-zero.[1] □

Then, Lemma 64 can be used to prove the following lemma.

Lemma 65 *For any $z \geq 1$, at least one column in the matrix product $\Pi_{t=u}^{u+rn-1} \mathbf{H}[t]$ is non-zero.*

Proof: Since $\Pi_{t=u}^{u+rn-1} \mathbf{H}[t]$ consists of rn connectivity matrices corresponding to reduced graphs, and the number of all reduced graphs is r, connectivity matrices corresponding to at least one reduced graph, say matrix $\mathbf{H}_*$, will appear in the above product at least n times.

Now observe that: (i) By Lemma 64, $\mathbf{H}_*^n$ contains a non-zero column, say the k-th column is non-zero, and (ii) by definition, all the $\mathbf{H}[t]$ matrices in the product contain a non-zero diagonal. These two observations together imply that the k-th column in the above product is non-zero.[2] □

For matrices $\mathbf{A}$ and $\mathbf{B}$ of identical dimension, we say that $\mathbf{A} \leq \mathbf{B}$ iff $\mathbf{A}_{ij} \leq \mathbf{B}_{ij}$ for all i, j. Lemma below relates the transition matrices with the connectivity matrices. Constant β used in the lemma below was introduced in Lemma 50.

Lemma 66 *For any $t \geq 1$, there exists a reduced graph $H[t] \in R_f$ such that $\beta\mathbf{H}[\mathbf{t}] \leq \mathbf{M}[t]$, where $\mathbf{H}[t]$ is the connectivity matrix for $H[t]$.*

Proof: First, let us construct a reduced graph $H[t]$: for each node i, removing a set of f node i's incoming links as defined in Lemma 50 (N_i^r). As a result, we have obtained a reduced graph $H[t]$ such that $\mathbf{M}_{ij}[t] \geq \beta$, if $j = i$ or edge (j, i) is in the reduced graph $H[t]$.

Denote by $\mathbf{H}[t]$ the connectivity matrix for the reduced graph $H[t]$. Then, $\mathbf{H}_{ij}[t]$ denotes the element in i-th row and j-th column of $\mathbf{H}[t]$. By definition of the connectivity matrix, we know that $\mathbf{H}_{ij}[t] = 1$, if $j = i$ or edge (j, i) is in the reduced graph; otherwise, $\mathbf{H}_{ij}[t] = 0$.

The statement in the lemma then follows from the above two observations. □

[1] That is, all the elements of the column will be non-zero. Also, such a non-zero column will exist in $\mathbf{H}^{n-1}[t]$, too. We use the loose bound of n to simplify the presentation.

[2] The product $\Pi_{t=u}^{u+rn-1} \mathbf{H}[t]$ can be viewed as the product of n instances of $\mathbf{H}_*$ "interspersed" with matrices with non-zero diagonals.

Let us now define a sequence of matrices $\mathbf{Q}(i)$, $i \geq 1$, such that each of these matrices is a product of rn of the $\mathbf{M}[t]$ matrices. Specifically,

$$\mathbf{Q}(i) \quad = \quad \Pi_{t=(i-1)rn+1}^{irn} \, \mathbf{M}[t] \tag{B.1}$$

From (7.7) and (B.1) observe that

$$v[krn] \quad = \quad \left(\Pi_{i=1}^{k} \, \mathbf{Q}(i) \right) \; v[0] \tag{B.2}$$

Based on (B.2), Lemmas 50, 65, and 66, we can show the following lemma.

Lemma 67 *For $i \geq 1$, $\mathbf{Q}(i)$ is a row stochastic matrix, and*

$$\lambda(\mathbf{Q}(i)) \leq 1 - \beta^{rn}.$$

Proof:

$\mathbf{Q}(i)$ is a product of row stochastic matrices ($\mathbf{M}[t]$); therefore, $\mathbf{Q}(i)$ is row stochastic. From Lemma 66, for each $t \geq 1$,

$$\beta \, \mathbf{H}[t] \;\leq\; \mathbf{M}[t]$$

Therefore,

$$\beta^{rn} \; \Pi_{t=(i-1)rn+1}^{irn} \; \mathbf{H}[t] \;\leq\; \Pi_{t=(i-1)rn+1}^{irn} \; \mathbf{M}[t] \;=\; \mathbf{Q}(i)$$

By using $u = (i-1)n + 1$ in Lemma 65, we conclude that the matrix product on the left side of the above inequality contains a non-zero column. Therefore, since $\beta > 0$, $\mathbf{Q}(i)$ on the right side of the inequality also contains a non-zero column.

Observe that rn is finite, and hence, β^{rn} is non-zero. Since the non-zero terms in $\mathbf{H}[t]$ matrices are all 1, the non-zero elements in $\Pi_{t=(i-1)rn+1}^{irn}\mathbf{H}[t]$ must each be ≥ 1. Therefore, there exists a non-zero column in $\mathbf{Q}(i)$ with all the elements in the column being $\geq \beta^{rn}$. Therefore, by Lemma 33, $\lambda(\mathbf{Q}(i)) \leq 1 - \beta^{rn}$, and $\mathbf{Q}(i)$ is a scrambling matrix. □

Let us now continue with the proof of ϵ-agreement. Consider the coefficient of ergodicity $\delta(\Pi_{u=1}^{t}\mathbf{M}[u])$.

$$
\begin{aligned}
\delta(\Pi_{u=1}^{t}\mathbf{M}[u]) &= \delta\left(\left(\Pi_{u=(\lfloor\frac{t}{rn}\rfloor)rn+1}^{t}\mathbf{M}[u]\right)\left(\Pi_{u=1}^{\lfloor\frac{t}{rn}\rfloor}\mathbf{Q}(u)\right)\right) \quad \text{by definition of } \mathbf{Q}(u) \\
&\leq \lambda\left(\Pi_{u=(\lfloor\frac{t}{rn}\rfloor)rn+1}^{t}\mathbf{M}[u]\right)\left(\Pi_{u=1}^{\lfloor\frac{t}{rn}\rfloor}\lambda\left(\mathbf{Q}(u)\right)\right) \quad \text{by Lemma 32} \\
&\leq \Pi_{u=1}^{\lfloor\frac{t}{rn}\rfloor}\lambda\left(\mathbf{Q}(u)\right) \quad \text{because } \lambda(\cdot)\leq 1 \\
&\leq (1-\beta^{rn})^{\lfloor\frac{t}{rn}\rfloor} \quad \text{by Lemma 46}
\end{aligned} \tag{B.3}
$$

Observe that the upper bound on right side of (B.3) depends only on graph $G(\mathcal{V},\mathcal{E})$ and t, and is independent of the input states, and the behavior of the faulty links. Moreover, the upper bound on the right side of (B.3) is a non-increasing function of t. Define t_{end} as the smallest positive integer such that the right hand side of (B.3) is smaller than $\frac{\epsilon}{n\max(|U|,|\mu|)}$. Recall that U and μ are defined as the upper and lower bound of the inputs at all nodes. Thus,

$$
\delta(\Pi_{u=1}^{t_{end}}\mathbf{M}[u]) \leq (1-\beta^{rn})^{\lfloor\frac{t_{end}}{rn}\rfloor} < \frac{\epsilon}{n\max(|U|,|\mu|)} \tag{B.4}
$$

Recall that β and r depend only on $G(\mathcal{V},\mathcal{E})$. Thus, t_{end} depends only on graph $G(\mathcal{V},\mathcal{E})$, and constants U,μ and ϵ.

By construction, $\Pi_{u=1}^{t}\mathbf{M}[u]$ is an $n\times n$ row stochastic matrix. Let $\mathbf{M}^*=\Pi_{u=1}^{t}\mathbf{M}[u]$. We omit time index $[t]$ from the notation $\mathbf{M}^*$ for simplicity. From (7.3), we have $v_j[t]=\mathbf{M}_j^* v[0]$. That is, the state of any node j can be obtained as the product of the j-th row of $\mathbf{M}^*$ and $v[0]$. Now, consider any two nodes j,k, we have

$$
\begin{aligned}
|v_j[t]-v_k[t]| &= |\mathbf{M}_j^* v[0]-\mathbf{M}_k^* v[0]| \\
&= |\Sigma_{i=1}^{n}\mathbf{M}_{ji}^* v_i[0]-\Sigma_{i=1}^{n}\mathbf{M}_{ki}^* v_i[0]| \\
&= |\Sigma_{i=1}^{n}\left(\mathbf{M}_{ji}^*-\mathbf{M}_{ki}^*\right)v_i[0]| \\
&\leq \Sigma_{i=1}^{n}|\mathbf{M}_{ji}^*-\mathbf{M}_{ki}^*||v_i[0]| \\
&\leq \Sigma_{i=1}^{n}\delta(\mathbf{M}^*)|v_i[0]| \\
&\leq n\delta(\mathbf{M}^*)\max(|U|,|\mu|) \\
&\leq n\delta(\Pi_{u=1}^{t}\mathbf{M}[u])\max(|U|,|\mu|)
\end{aligned} \tag{B.5}
$$

Therefore, by (B.4) and (B.5), we have

$$|v_j[t_{end}] - v_k[t_{end}]| < \epsilon \tag{B.6}$$

Since the output of the nodes equal its state at termination (after t_{end} iterations). Thus, (B.6) implies that Algorithm 4 satisfies the ϵ-agreement property.

References

[1] Cassandra. `http://cassandra.apache.org/`.

[2] I. Abraham, Y. Amit, and D. Dolev. Optimal resilience asynchronous approximate agreement. In *OPODIS*, pages 229–239, 2004.

[3] E. Alchieri, A. Bessani, J. Silva Fraga, and F. Greve. Byzantine consensus with unknown participants. In T. Baker, A. Bui, and S. Tixeuil, editors, *Principles of Distributed Systems*, volume 5401 of *Lecture Notes in Computer Science*, pages 22–40. Springer Berlin Heidelberg, 2008.

[4] H. Attiya, A. Bar-Noy, D. Dolev, D. Peleg, and R. Reischuk. Renaming in an asynchronous environment. *Journal of the ACM*, July 1990.

[5] H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations, and Advanced Topics.* Wiley Series on Parallel and Distributed Computing, 2004.

[6] A. Azadmanesh and H. Bajwa. Global convergence in partially fully connected networks (pfcn) with limited relays. In *Industrial Electronics Society, 2001. IECON '01. The 27th Annual Conference of the IEEE*, volume 3, pages 2022 –2025 vol.3, 2001.

[7] M. H. Azadmanesh and R. Kieckhafer. Asynchronous approximate agreement in partially connected networks. *International Journal of Parallel and Distributed Systems and Networks*, 5(1):26–34, 2002.

[8] P. Bansal, P. Gopal, A. Gupta, K. Srinathan, and P. K. Vasishta. Byzantine agreement using partial authentication. In *Proceedings of the 25th international conference on Distributed computing*, DISC'11, pages 389–403, Berlin, Heidelberg, 2011. Springer-Verlag.

[9] D. P. Bertsekas and J. N. Tsitsiklis. *Parallel and Distributed Computation: Numerical Methods.* Optimization and Neural Computation Series. Athena Scientific, 1997.

[10] V. Bhandari and N. H. Vaidya. On reliable broadcast in a radio network. In *Proceedings of the twenty-fourth annual ACM symposium on Principles of distributed computing*, PODC '05, pages 138–147, New York, NY, USA, 2005. ACM.

[11] V. Bhandari and N. H. Vaidya. On reliable broadcast in a radio network: A simplified characterization. Technical report, University of Illinois at Urbana-Champaign, 2005.

[12] M. Biely, P. Robinson, and U. Schmid. Agreement in directed dynamic networks. In *Structural Information and Communication Complexity*, volume 7355 of *Lecture Notes in Computer Science*, pages 73–84. Springer Berlin Heidelberg, 2012.

[13] M. Biely, P. Robinson, U. Schmid, M. Schwarz, and K. Winkler. Gracefully degrading consensus and k-set agreement in directed dynamic networks. *CoRR*, abs/1408.0620, 2014.

[14] M. Biely, U. Schmid, and B. Weiss. Synchronous consensus under hybrid process and link failures. *Theor. Comput. Sci.*, 412(40):5602–5630, Sept. 2011.

[15] M. Burrows. The chubby lock service for loosely-coupled distributed systems. In *Proceedings of the 7th Symposium on Operating Systems Design and Implementation*, OSDI '06, pages 335–350, Berkeley, CA, USA, 2006. USENIX Association.

[16] B. Calder, J. Wang, A. Ogus, N. Nilakantan, A. Skjolsvold, S. McKelvie, Y. Xu, S. Srivastav, J. Wu, H. Simitci, J. Haridas, C. Uddaraju, H. Khatri, A. Edwards, V. Bedekar, S. Mainali, R. Abbasi, A. Agarwal, M. F. u. Haq, M. I. u. Haq, D. Bhardwaj, S. Dayanand, A. Adusumilli, M. McNett, S. Sankaran, K. Manivannan, and L. Rigas. Windows azure storage: A highly available cloud storage service with strong consistency. In *Proceedings of the Twenty-Third ACM Symposium on Operating Systems Principles*, SOSP '11, pages 143–157, New York, NY, USA, 2011. ACM.

[17] B. Charron-Bost, M. Függer, and T. Nowak. Approximate consensus in highly dynamic networks. *CoRR*, abs/1408.0620, 2014.

[18] B. Charron-Bost, M. Függer, and T. Nowak. Approximate consensus in highly dynamic networks: The role of averaging algorithms. In *Automata, Languages, and Programming - 42nd International Colloquium, ICALP 2015, Kyoto, Japan, July 6-10, 2015, Proceedings, Part II*, pages 528–539, 2015.

[19] B. Charron-Bost and A. Schiper. The heard-of model: computing in distributed systems with benign faults. *Distributed Computing*, 22(1):49–71, 2009.

[20] B. A. Coan. A compiler that increases the fault tolerance of asynchronous protocols. *IEEE Trans. Comput.*, 37(12):1541–1553, Dec. 1988.

[21] B. F. Cooper, R. Ramakrishnan, U. Srivastava, A. Silberstein, P. Bohannon, H.-A. Jacobsen, N. Puz, D. Weaver, and R. Yerneni. Pnuts: Yahoo!'s hosted data serving platform. *PVLDB*, 1(2):1277–1288, 2008.

[22] J. C. Corbett et al. Spanner: Google's globally-distributed database. In *Proc. USENIX Conference on Operating Systems Design and Implementation (OSDI)*, pages 251–264, 2012.

[23] M. Correia, D. G. Ferro, F. P. Junqueira, and M. Serafini. Practical hardening of crash-tolerant systems. In *Proceedings of the 2012 USENIX Conference on Annual Technical Conference*, USENIX ATC'12, pages 41–41, Berkeley, CA, USA, 2012. USENIX Association.

[24] S. Dasgupta, C. Papadimitriou, and U. Vazirani. *Algorithms.* McGraw-Hill Higher Education, 2006.

[25] G. DeCandia, D. Hastorun, M. Jampani, G. Kakulapati, A. Lakshman, A. Pilchin, S. Sivasubramanian, P. Vosshall, and W. Vogels. Dynamo: Amazon's highly available key-value store. In *Proc. ACM SIGOPS Symposium on Operating Systems Principles (SOSP)*, pages 205–220, 2007.

[26] Y. Desmedt and Y. Wang. Perfectly secure message transmission revisited. In L. Knudsen, editor, *Advances in Cryptology – EUROCRYPT 2002*, volume 2332 of *Lecture Notes in Computer Science*, pages 502–517. Springer Berlin Heidelberg, 2002.

[27] D. Dolev. The Byzantine generals strike again. *Journal of Algorithms*, 3(1), March 1982.

[28] D. Dolev, C. Dwork, O. Waarts, and M. Yung. Perfectly secure message transmission. *Journal of the Association for Computing Machinery (JACM)*, 40(1):17–14, 1993.

[29] D. Dolev, N. A. Lynch, S. S. Pinter, E. W. Stark, and W. E. Weihl. Reaching approximate agreement in the presence of faults. *J. ACM*, 33:499–516, May 1986.

[30] D. Easley and J. Kleinberg. *Networks, Crowds, and Markets: Reasoning about a Highly Connected World.* Cambridge, 2010.

[31] A. D. Fekete. Asymptotically optimal algorithms for approximate agreement. In *Proceedings of the fifth annual ACM symposium on Principles of distributed computing*, PODC '86, pages 73–87, New York, NY, USA, 1986. ACM.

[32] M. J. Fischer, N. A. Lynch, and M. Merritt. Easy impossibility proofs for distributed consensus problems. In *Proceedings of the fourth annual ACM symposium on Principles of distributed computing*, PODC '85, pages 59–70, New York, NY, USA, 1985. ACM.

[33] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of distributed consensus with one faulty process. *J. ACM*, 32:374–382, April 1985.

[34] J. Hajnal. Weak ergodicity in non-homogeneous Markov chains. In *Proceedings of the Cambridge Philosophical Society*, volume 54, pages 233–246, 1958.

[35] M. Herlihy, D. Kozlov, and S. Rajsbaum. *Distributed Computing Through Combinatorial Topology.* Elsevier Science, 2013.

[36] D. Huttenlocher, G. Klanderman, and W. Rucklidge. Comparing images using the Hausdorff distance. *Pattern Analysis and Machine Intelligence, IEEE Transactions on*, 15(9):850–863, 1993.

[37] A. Ichimura and M. Shigeno. A new parameter for a broadcast algorithm with locally bounded Byzantine faults. *Inf. Process. Lett.*, June 2010.

[38] A. Jadbabaie, J. Lin, and A. Morse. Coordination of groups of mobile autonomous agents using nearest neighbor rules. *Automatic Control, IEEE Transactions on*, 48(6):988 – 1001, june 2003.

[39] F. Junqueira and K. Marzullo. Synchronous consensus for dependent process failures. In *Distributed Computing Systems, 2003. Proceedings. 23rd International Conference on*, pages 274–283, May 2003.

[40] D. Kempe, A. Dobra, and J. Gehrke. Gossip-based computation of aggregate information. pages 482–491. IEEE Computer Society, 2003.

[41] R. M. Kieckhafer and M. H. Azadmanesh. Low cost approximate agreement in partially connected networks. *Journal of Computing and Information*, 3(1):53–85, 1993.

[42] C.-Y. Koo. Broadcast in radio networks tolerating Byzantine adversarial behavior. In *Proceedings of the twenty-third annual ACM symposium on Principles of distributed computing*, PODC '04, pages 275–282, New York, NY, USA, 2004. ACM.

[43] P. Kuznetsov. Understanding non-uniform failure models. *Bulletin of the European Association for Theoretical Computer Science (BEATCS)*, 106:53–77, 2012.

[44] A. Lakshman and P. Malik. Cassandra: a decentralized structured storage system. *ACM SIGOPS Operating Systems Review*, 44(2):35–40, 2010.

[45] L. Lamport. The part-time parliament. *ACM Trans. Comput. Syst.*, 16(2):133–169, May 1998.

[46] L. Lamport, R. Shostak, and M. Pease. The Byzantine generals problem. *ACM Trans. on Programming Languages and Systems*, 1982.

[47] H. LeBlanc and X. Koutsoukos. Consensus in networked multi-agent systems with adversaries. *14th International conference on Hybrid Systems: Computation and Control (HSCC)*, 2011.

[48] H. LeBlanc and X. Koutsoukos. Low complexity resilient consensus in networked multi-agent systems with adversaries. *15th International conference on Hybrid Systems: Computation and Control (HSCC)*, 2012.

[49] H. LeBlanc, H. Zhang, X. Koutsoukos, and S. Sundaram. Resilient asymptotic consensus in robust networks. *IEEE Journal on Selected Areas in Communications: Special Issue on In-Network Computation*, 31:766–781, April 2013.

[50] H. LeBlanc, H. Zhang, S. Sundaram, and X. Koutsoukos. Consensus of multi-agent networks in the presence of adversaries using only local information. *HiCoNs*, 2012.

[51] D. S. Lun, M. MéDard, R. Koetter, and M. Effros. Full length article: On coding for reliable communication over packet networks. *Phys. Commun.*, 1(1):3–20, Mar. 2008.

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

[53] A. Maurer, S. Tixeuil, and X. Défago. Reliable communication in a dynamic network in the presence of Byzantine faults. *CoRR*, abs/1402.0121, 2014.

[54] H. Mendes and M. Herlihy. Multidimensional approximate agreement in Byzantine asynchronous systems. In *STOC '13*, 2013.

[55] H. Mendes, C. Tasson, and M. Herlihy. Brief announcement: The topology of asynchronous byzantine colorless tasks. In *The 27th International Symposium on Distributed Computing (DISC)*, 2013.

[56] R. Olfati-Saber, J. Fax, and R. Murray. Consensus and cooperation in networked multi-agent systems. *Proceedings of the IEEE*, 95(1):215–233, Jan 2007.

[57] A. Pagourtzis, G. Panagiotakos, and D. Sakavalas. Reliable broadcast with respect to topology knowledge. In *Proceedings of the 28th international conference on Distributed computing (DISC)*, 2014.

[58] M. Pajic, S. Sundaram, J. Le Ny, G. J. Pappas, and R. Mangharam. Closing the loop: A simple distributed method for control over wireless networks. In *Proceedings of the 11th International Conference on Information Processing in Sensor Networks*, IPSN '12, pages 25–36, New York, NY, USA, 2012. ACM.

[59] L. Parker. Current state of the art in distributed autonomous mobile robotics. In L. Parker, G. Bekey, and J. Barhen, editors, *Distributed Autonomous Robotic Systems 4*, pages 3–12. Springer Japan, 2000.

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

[61] A. Pelc and D. Peleg. Broadcasting with locally bounded Byzantine faults. *Inf. Process. Lett.*, 2005.

[62] M. A. Perles and M. Sigorn. A generalization of Tverberg's theorem. *CoRR*, abs/0710.4668, 2007.

[63] M. Rubenstein, C. Ahler, N. Hoff, A. Cabrera, and R. Nagpal. Kilobot: A low cost robot with scalable operations designed for collective behaviors. *Robotics and Autonomous Systems*, 62(7):966–975, 2014.

[64] N. Santoro and P. Widmayer. Time is not a healer. In *Proceedings of the 6th Annual Symposium on Theoretical Aspects of Computer Science on STACS 89*, pages 304–313, New York, NY, USA, 1989. Springer-Verlag New York, Inc.

[65] N. Santoro and P. Widmayer. Agreement in synchronous networks with ubiquitous faults. *Theor. Comput. Sci.*, 384(2-3):232–249, Oct. 2007.

[66] I. Schizas, A. Ribeiro, and G. Giannakis. Consensus in ad hoc WSNs with noisy links – Part I: Distributed estimation of deterministic signals. *Signal Processing, IEEE Transactions on*, 56(1):350–364, Jan 2008.

[67] U. Schmid, B. Weiss, and I. Keidar. Impossibility results and lower bounds for consensus under link failures. *SIAM J. Comput.*, 38(5):1912–1951, Jan. 2009.

[68] F. B. Schneider. Implementing fault-tolerant services using the state machine approach: A tutorial. *ACM Comput. Surv.*, 22(4):299–319, Dec. 1990.

[69] M. Schwarz, K. Winkler, U. Schmid, M. Biely, and P. Robinson. Brief announcement: Gracefully degrading consensus and k-set agreement under dynamic link failures. In *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing*, PODC '14, pages 341–343, New York, NY, USA, 2014. ACM.

[70] N. B. Shah, K. V. Rashmi, and K. Ramchandran. Secret share dissemination across a network. *CoRR*, abs/1207.0120, 2012.

[71] B. Shankar, P. Gopal, K. Srinathan, and C. P. Rangan. Unconditionally reliable message transmission in directed networks. In *Proceedings of the nineteenth annual ACM-SIAM symposium on Discrete algorithms*, SODA '08, pages 1048–1055, Philadelphia, PA, USA, 2008. Society for Industrial and Applied Mathematics.

[72] L. Su and N. Vaidya. Reaching approximate Byzantine consensus with multi-hop communication. In A. Pelc and A. A. Schwarzmann, editors, *Stabilization, Safety, and Security of Distributed Systems*, volume 9212 of *Lecture Notes in Computer Science*, pages 21–35. Springer International Publishing, 2015.

[73] S. Sundaram and C. Hadjicostis. Distributed function calculation and consensus using linear iterative strategies. *Selected Areas in Communications, IEEE Journal on*, 26(4):650–660, May 2008.

[74] S. Sundaram, S. Revzen, and G. Pappas. A control-theoretic approach to disseminating values and overcoming malicious links in wireless networks. *Automatica*, 48(11):2894–2901, Nov. 2012.

[75] L. Tseng, N. Vaidya, and V. Bhandari. Broadcast using certified propagation algorithm in presence of Byzantine faults. *Information Processing Letters*, 115(4):512 – 514, 2015.

[76] L. Tseng and N. H. Vaidya. Exact Byzantine consensus in directed graphs. *CoRR*, abs/1208.5075, 2012.

[77] L. Tseng and N. H. Vaidya. Byzantine convex consensus: An optimal algorithm. *CoRR*, abs/1307.1332, 2013.

[78] L. Tseng and N. H. Vaidya. Asynchronous convex consensus in the presence of crash faults. *CoRR*, abs/1403.3455, 2014.

[79] L. Tseng and N. H. Vaidya. Asynchronous convex hull consensus in the presence of crash faults. In *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing*, PODC '14, pages 396–405, New York, NY, USA, 2014. ACM.

[80] L. Tseng and N. H. Vaidya. Crash-tolerant consensus in directed graphs. *CoRR*, abs/1412.8532, 2014.

[81] L. Tseng and N. H. Vaidya. Iterative approximate consensus in the presence of Byzantine link failures. In *Networked Systems - Second International Conference, NETYS 2014, Marrakech, Morocco, May 15-17, 2014. Revised Selected Papers*, pages 84–98, 2014.

[82] L. Tseng and N. H. Vaidya. Iterative approximate consensus in the presence of Byzantine link failures. *CoRR*, Jan. 2014.

[83] L. Tseng and N. H. Vaidya. Fault-tolerant consensus in directed graphs. In *Proceedings of the 2015 ACM Symposium on Principles of Distributed Computing*, PODC '15, pages 451–460, New York, NY, USA, 2015. ACM.

[84] L. Tseng and N. H. Vaidya. Iterative approximate Byzantine consensus under a generalized fault model. In *In International Conference on Distributed Computing and Networking (ICDCN)*, January 2013.

[85] L. Tseng, N. H. Vaidya, and V. Bhandari. Broadcast using certified propagation algorithm in presence of Byzantine faults. *CoRR*, abs/1209.4620, 2012.

[86] N. H. Vaidya. Matrix representation of iterative approximate Byzantine consensus in directed graphs. *CoRR*, Mar. 2012.

[87] N. H. Vaidya. Iterative Byzantine vector consensus in incomplete graphs. In *In International Conference on Distributed Computing and Networking (ICDCN)*, January 2014.

[88] N. H. Vaidya and V. K. Garg. Byzantine vector consensus in complete graphs. In *Proceedings of the 2013 ACM Symposium on Principles of Distributed Computing*, PODC '13, pages 65–73, New York, NY, USA, 2013. ACM.

[89] N. H. Vaidya, L. Tseng, and G. Liang. Iterative approximate Byzantine consensus in arbitrary directed graphs. In *Proceedings of the thirty-first annual ACM symposium on Principles of distributed computing*, PODC '12. ACM, 2012.

[90] N. H. Vaidya, L. Tseng, and G. Liang. Iterative approximate Byzantine consensus in arbitrary directed graphs. *CoRR*, abs/1201.4183, 2012.

[91] N. H. Vaidya, L. Tseng, and G. Liang. Iterative approximate Byzantine consensus in arbitrary directed graphs - part ii: Synchronous and asynchronous systems. *CoRR*, abs/1202.6094, 2012.

[92] D. B. West. *Introduction To Graph Theory.* Prentice Hall, 2001.

[93] J. Wolfowitz. Products of indecomposable, aperiodic, stochastic matrices. In *Proceedings of the American Mathematical Society*, volume 14, pages 733–737, 1963.

[94] H. Zhang and S. Sundaram. Robustness of complex networks with implications for consensus and contagion. In *Proceedings of CDC 2012, the 51st IEEE Conference on Decision and Control*, 2012.