DISS. ETH NO. 26959

Byzantine Agreement on Representative Input Values Over Public Channels

A thesis submitted to attain the degree of

DOCTOR OF SCIENCES

(Dr. sc. ETH Zurich)

presented by

DARYA MELNYK

M.Sc.(TUM)
born on 05.10.1991
citizen of Germany

accepted on the recommendation of

Prof. Dr. Roger Wattenhofer, examiner
Dr. Ittai Abraham, co-examiner
Prof. Dr. Bryan Ford, co-examiner

2020

First Edition 2020

Series in Distributed Computing, Volume 33
TIK-Schriftenreihe-Nr. 185
Edited by Roger Wattenhofer

Free Space Publishing
ISBN 9798563171725

Abstract

In the Byzantine agreement problem, n nodes with possibly different input values aim to reach agreement on a common value in the presence of $t < n/3$ Byzantine parties by communicating over a fully connected network. This dissertation focuses on a special communication model, called the *public channel* model, where the adversary can see all values sent over the channels of the network. In the first part of the dissertation, the synchronous message passing model is considered. It is assumed that the values of the nodes are ordered and a novel validity condition that accepts consensus values that are close to the k^{th} smallest value of the correct nodes is introduced. A deterministic algorithm is proposed in order to approximate the k^{th} smallest value. It is then shown that this approximation is the best possible. In the next step, a multidimensional case is considered. In this case, the input values of the nodes are assumed to be preference rankings of three or more candidates. This so-called preferential voting raises new questions about how to approximate consensus vectors. At first, a deterministic algorithm to solve Byzantine agreement on rankings is proposed under a generalized validity condition, which will be called Pareto validity. These results are then extended by considering a special voting rule which chooses the Kemeny median as the consensus vector.

This dissertation also considers the shared memory model as a possible communication model for blockchain protocols. It presents a novel shared memory model that simplifies the analysis of consensus on a Chain and a DAG. In this new model, referred to as the append memory model, nodes are allowed to write their values to the unordered memory, but not to overwrite already existing values. It is shown that although this model differs from the standard shared memory model with n shared read-write registers, many known results from the shared memory model still hold in the append

memory model. Assuming a probabilistic access to the append memory, the Byzantine agreement protocols on the Chain and the DAG are compared. It is shown that the DAG structure achieves an almost optimal resilience (close to $t < n/2$) in contrast to the Chain structure whose resilience depends on the access rate to the memory.

Another considered communication model is the asynchronous message-passing model. In this part, the idea of the blackboard model is used to define a strong broadcast abstraction called blackboard broadcast. Since the nodes receive almost the same input values from all other nodes when the blackboard broadcast is used, detectability of Byzantine behavior can be defined with respect to their input values. It is shown that Byzantine behavior is not easily detectable in this model if message scheduling and $t \in \Omega(n)$ input values are controlled by the Byzantine party. To develop efficient Byzantine agreement algorithms in the asynchronous communication model, a deep reinforcement learning framework (DRL) is implemented. It is shown that DRL can be used to learn Byzantine behavior. To extend this, a self-play environment is considered, where a Byzantine agent is competing against a collection of correct agents. The results show that DRL can in general be applied to standard Byzantine agreement problems and that Byzantine behavior is tentatively harder to learn when the correct nodes use fully randomized strategies.

Zusammenfassung

Das Problem der byzantinischen Generäle ist ein Kommunikationsproblem, bei dem n Prozessoren, welche verschiedene Eingabewerte besitzen, sich auf einen Wert einigen. Die Prozessoren kommunizieren dabei über ein vollständig verbundenes Netzwerk, und es wird angenommen, dass $t < n/3$ der Prozessoren byzantinischen Verhalten aufweisen. Im ersten Teil dieser Dissertation wird das synchrone Message-Passing-Modell betrachtet. Es wird angenommen, dass die Eingabewerte geordnet werden können. Unter dieser Annahme wird eine Validität-Bedingung vorgestellt, die eine Übereinkunft auf einem Wert zulässt, der nahe des k-ten Wertes aller geordneten Eingabewerte liegt. Es wird eine Methode vorgestellt, die es ermöglicht diese Validität-Bedingung zu erfüllen. Im nächsten Schritt werden mehrdimensional Eingabewerte betrachtet. In diesem Fall wird angenommen, dass die Eingabewerte aus Präferenz-Rankings von 3 oder mehr Kandidaten bestehen. Das sogenannte Präferenzwahlsystem wirft neue Fragen darüber auf, wie eine faires Konsensus-Ranking unter der Annahme von byzantinischen Prozessoren im System approximiert werden kann. Es wird ein deterministischer Algorithmus vorgestellt, der das Problem der byzantinischen Generäle für Rankings als Eingabewerte löst unter der Annahme einer verallgemeinerten Validität-Bedingung – der Pareto-Validität. Diese Ergebnisse werden verwendet, um ein spezielles Wahlsystem zu betrachten – das der Approximation eines Kemeny-Medians als Konsensus-Ranking.

Diese Dissertation betrachtet weiter das Shared-Memory-Modell als ein mögliches Kommunikationssystem für Blockchainsysteme. Es wird ein neuartiges Shared-Memory-Modell vorgestellt, welches die Analysis von Konsensus-Protokollen für die Chain- und die DAG-Strukturen vereinfacht. In diesem Modell, welches als das Append-Memory-Modell bezeichnet wird, dürfen die Prozessoren ihre Werte in einen ungeordneten Speicher schrei-

ben, jedoch die dort bereits vorhandenen Werte nicht ersetzen. Es wird gezeigt, dass, obwohl dieses Modell sich vom Shared-Memory-Modell unterscheidet, viele bekannte Resultate aus dem Shared-Memory-Modell auf das Append-Memory-Modell übertragen lassen. Unter der Annahme, dass der Zugriff zum Append-Speicher probabilistisch ist, werden verschiedene Protokolle für das Problem der byzantinischen Generäle auf der Chain- und der DAG-Struktur verglichen. Es stellt sich heraus, dass die DAG-Struktur die optimale Anzahl von $t < n/2$ byzantinischen Prozessoren tolerieren kann, während die Anzahl byzantinischer Prozessoren, welche die Chain-Struktur tolerieren kann, indirekt proportional zur Zugriffsrate auf den Speicher ist.

Ein weiteres Kommunikationsmodell, das in dieser Arbeit betrachtet wird, ist das asynchrone Message-Passing-Modell. In diesem Teil der Arbeit wird die Idee des Blackboard-Modells verwendet, um ein zuverlässigeres Übertragungsprotokoll einzuführen, den sogenannten Blackboard-Broadcast. Da die Prozessoren unter der Annahme des Blackboard-Broadcast bei der Kommunikation fast identische Werte erhalten, kann byzantinisches Verhalten durch das Betrachten der Werte nachgewiesen werden. Es wird gezeigt, dass es dennoch nicht einfach möglich ist byzantinisches Verhalten in diesem Modell nachzuweisen, sofern die Ankunftszeit der Nachrichten sowie $t \in \Omega(n)$ Prozessoren von einem byzantinischen Gegner kontrolliert werden. Um eine effiziente Lösung für das Problem der byzantinischen Generäle zu finden, wird ein Deep Reinforcement Learning (DRL) Framework implementiert. Es wird zunächst bestätigt, dass DRL sich eignet, um byzantinisches Verhalten zu erlernen. Daraufhin wird dieses Resultat im Self-Play-Scenario betrachtet, bei dem byzantinische Agenten gegen eine Gruppe von korrekten Agenten antreten. Die Ergebnisse aus dieser Arbeit zeigen, dass DRL sich auf das Problem der byzantinischen Generäle anwenden lässt und, dass byzantinisches Verhalten schwieriger zum Lernen ist, wenn die korrekten Prozessoren randomisierte Strategien verwenden können.

Acknowledgments

During my time as a PhD student in the Distributed Computing Group (DISCO) at ETH Zurich, I have had the opportunity to develop versatile skills: I have learned how to supervise students, prepare lecture notes, give great talks and, most importantly, how to carry out research. All of this would not have been possible without the guidance of my supervisor, Prof. Roger Wattenhofer. I would like to thank him for the opportunity to pursue a PhD in his group, for his encouragement to look into different topics beyond my main expertise, for sharing his devotion to teaching and for keeping up the tradition of weekly meetings.

I would also like to thank my co-referees Ittai Abraham and Bryan Ford for taking the time to read my thesis, providing me with constructive feedback and attending my defense.

In the past four years, I had the opportunity to work with wonderful colleagues who contributed to my research with their great ideas, their willingness to serve as test persons in my projects, and their jolly spirit. I have especially enjoyed our Töggeli-traditions, hiking trips and ice skating events, as well as simply having such nice people to talk to. I would therefore like to thank each one of you in alphabetical order:

Aryaz Eghbali for preferring sledding over skiing and ordering a Rösti pizza at the restaurant; Beat Futterknecht for being able to answer every question I had outside of research, for being a great listener and giving great advice; Béni Egressy for teaching me about British culture; Christian Fluri for enabling me to visit the Iguazú Falls; Conrad Burchert for sharing his love for the blockchain protocols; Damian Pascual for the stories from the not so safe adventures in the Swiss alps; Friederike Bruetsch for always providing us with a piece of chocolate in her office; Georg Bachmeier for his determination to learn how to bake and all the tasty cakes that followed; Gino Brunner for

being the only one afraid to play against me at Töggeli despite being the best player, as well as for being the expert on every gadget that you might ever need; Henri Devillez for introducing me to the art of cubing and for having the most exquisite cube collection in the office; Jakub Sliwinski for being in the office at random hours; Julian Steger for teaching me how to take care of our office plants; Klaus-Tycho Förster for giving me valuable advice for my future plans and always offering a helping hand; Ladislas Jacobe de Naurois for showing how to give a great talk without slides; Laura Peer for being a great Töggeli partner; Lukas Faber for encouraging people to bake more cakes in my lasts few months than we have had in the past two years; Manuel Eichelberger for finding a simple and useless solution to every problem in life and for prolonging our hike all the way to his orchestra practice; Michael König for his hidden talent at sports and his unique healthy diet; Oliver Richter for being my fist seminar student; Pál András Papp for teaching me how to pronounce "Kemeny" correctly, for drawing colorful forests on many whiteboards and for his good humor; Pankaj Khanchandani for his lessons on how to differentiate between "d" and "d" and for knowing every single person in distributed computing; Pascal Bissig for his alternative humor; Philipp Brandes for being a good company during our apple breaks, for being the first colleague to join on a hike through Swiss mountains and for being a good friend; Roland Schmid for his ability to decorate the office with just one plant; Sebastian Brandt for teaching me basic Töggeli shots; Simon Tanner for being a great companion on many hikes, skiing trips, climbs and cave excursions, for keeping alive a constant flow of students to our office and for sharing a passion for plants; Susann Arreghini for all the support during the exam sessions and for many interesting discussions; Tejaswi Nadahalli for his amazing programming skills and for sharing the best Swiss chocolate; Thomas Ulrich for being a very good friend and a great office mate, for saving all my plants during tough times and for sharing Thursday evenings at the gym; Ye Wang for always having some sweets or vouchers to share and for hoarding essential foods in the office; Yuyi Wang for being my first office mate, for introducing me to new students all the time and for keeping alive a giant network of researchers while traveling the world; Zeta Avarikioti for hosting great rooftop parties with exquisite views; Zhao Meng for treating us with a delicious green tea cake.

This thesis would not have been possible to complete without the strong support of my family. I would especially like to thank my grandma Lena for being my role model and teaching me that a woman can achieve anything she wants. I also want to thank my parents Yuriy and Inna for always supporting my dream to become a researcher, even though a dentist in the family would have been an asset. I want to thank my brother Sergiy for always challenging and supporting me. And I would like to thank Daniel Lindblad for always

being the first to proofread my papers, for taking countless trips to Zurich and simply for being my very best friend.

Finally, I would like to thank all my friends for the support they gave me during my PhD and for visiting me in Zurich during this time.

Collaborations and Contributions

This dissertation resulted from a collaboration with colleagues whom I would like to thank for the fruitful discussions and their contribution to the projects. Below, the collaborators that contributed to the single chapters of this dissertation are listed in alphabetical order. The referenced work listed here is joint work with my supervisor Roger Wattenhofer.

Chapter 2 is based on Chapters 4 and 6 of the book *Blockchain Science – Distributed Ledger Technology* [123]. Co-author is Barbara Keller.

Chapter 3 is based on the publication *Byzantine Agreement with Interval Validity* [84].

Chapter 4 is based on the publication *Byzantine Preferential Voting* [82]. Co-author is Yuyi Wang.

Chapter 5 is based on the publication *The Append Memory Model: Why BlockDAGs Excel Blockchains* [85].

Chapter 6, Section 6.1 is based on the publication draft *Towards the Impossibility of a Byzantine Shared Coin* [83]. Co-author is Yuyi Wang.

Chapter 6, Section 6.2 and 6.3 are based on the publication draft *Asynchronous Byzantine Agreement with Reinforcement Learning* [81]. Co-authors are Janka Möller, Lazar Rakic and Oliver Richter.

Contents

1

Introduction

> *"It's amazing how clever people can be, but when you build a new system it is very, very hard to imagine the ways in which it can be attacked."*
>
> — Sir Tim Berners-Lee, founder of the World Wide Web

Distributed systems have many advantages over their centralized counterparts. Consider for example a central bank: if the central server of this bank is attacked, it is not possible to make any payments until the problem is fixed. In a distributed banking system, intact peers can restore the banking details and allow transactions even in the case of an attack. Another example is the safety of self-driving cars. With a single sensor, the reading of the measured value can be unreliable or even completely wrong. To improve the accuracy of the measured value, a larger number of sensors can be used. Distributed systems also have the potential of ensuring data privacy: in case of a virus-tracking app, it is advisable to use a distributed version that stores the information about interactions with other people on a local device, instead of revealing it to the government. The presented examples profit from many attractive properties of distributed systems: There exists no trusted central authority and thus no single point of failure; the data can

be reconstructed by different parties and is, therefore, more reliable and; it is possible to solve tasks without sharing your personal data with the peers.

The absence of a trusted authority introduces challenges when implementing such a distributed system. Multiple participants need to communicate with each other and make decisions to solve a common task. The problem of deciding on a common value in a distributed system is called consensus. Some of the key results in distributed computing promise to provide a distributed system that can tolerate arbitrary failures. Such failures are usually called *Byzantine*. The machines (nodes) of a distributed system regularly check if they are in the same state. Whenever the nodes disagree, they run a *Byzantine agreement* protocol to eliminate blunders by some nodes and thus enforce agreement. If nodes continuously agree on their state, the distributed system as a whole is correct.

For a distributed system to be applicable in practice, a Byzantine agreement protocol should be fast and ensure that all nodes agree on a common state. These criteria are known as the *termination time* and the *agreement property*, respectively. However, the two properties are not enough. For instance, a protocol may just agree to delete all the information in the system, fulfilling both termination and agreement, but potentially also destroying valuable data. To prevent such absurd "solutions", we need a third property, known as the *validity property*. Informally, the validity property must make sure that the decision "makes sense". In particular, if all nodes of a distributed system propose the same state, they should settle for that state.

Many of the proposed solutions for Byzantine agreement rely on cryptography by either considering computationally bounded adversaries or use homomorphic encryption in the information theoretic setting. Encryption schemes require more computational power from the peers and potentially introduce new points of failure to the system. In this dissertation, we therefore consider a computationally unbounded adversary and even assume that the Byzantine adversary can see the information that is transmitted over all communication channels. It has been shown that it is impossible to solve Byzantine agreement in such a system if one third of the participants are Byzantine. We therefore assume that less than a third of the parties are Byzantine and focus on deriving meaningful validity conditions. In particular, we want to see how reliable distributed systems can be if they have to agree on real values, such as on a true sensor value in a car. We also go a step further and consider a multi-dimensional setting where the decision must be based on rankings. Such applications can for example be found in large-scale machine learning systems, where a final decision has to be drawn from a set of rankings. This work shows that while Byzantine nodes can prevent the systems from agreeing on the true value, it is still possible to agree on a fairly good value or ranking.

Besides validity conditions, there is a need in developing transparent models for distributed agreement algorithms. Agreement algorithms are usually deployed as subroutines and make it possible to reliably solve more complicated tasks. A complicated model for agreement might lead to systems that make false assumptions on their subroutines. In this spirit, we develop a novel model for analyzing blockchain protocols that is based on a shared memory instead of a peer-to-peer communication. This model can be simulated in a peer-to-peer setting and satisfies the same properties. The advantage of the shared memory is that it becomes more transparent why no asynchronous Byzantine agreement is possible in the blockchain when the adversary is computationally unbounded. In such way, restricted communication allows us to derive a simpler correctness analysis of blockchain protocols and compare them against each other. We further consider the asynchronous Byzantine agreement model, where the messages between the peers of a system can be arbitrarily delayed. Our analysis shows that complicated Byzantine agreement algorithms tend to make an analysis of all conceivable Byzantine strategies nearly impossible. We therefore investigate deep reinforcement learning methods to understand Byzantine behavior and could enable the development of simple and efficient Byzantine agreement algorithms in the future.

2

Preliminaries

In this chapter, the main concepts of Byzantine agreement in the message passing and shared memory models that will be used throughout the dissertation will be presented. This includes the concepts of synchronous and asynchronous communications systems, various validity conditions used to determine the quality of the consensus value, and the differences between the shared memory and message passing models. Some selected algorithms from the literature that will help the reader understand the algorithms discussed in this dissertation will also be presented. After the fundamentals have been covered, the main results of each chapter in this thesis, together with related work, will be provided in Section 2.2.

2.1 Introduction to Byzantine Agreement

In this work, we consider Byzantine agreement (BA) in a distributed system of n nodes. This problem was originally introduced as the "Byzantine Generals Problem" by Pease, Shostak, and Lamport [72, 100]. In this problem, every node has a unique ID $i \in [n]$ and an input value v_i given to it at the beginning of the algorithm. The nodes can either communicate with each other over an all-to-all network via authenticated channels or via a common memory. These communication types describe the message-passing

and shared memory models. We assume that there are no network failures and that if a message is sent via an authenticated channel or written to the memory - the receiver always knows the sender. The goal of the algorithm is to establish consensus on a common value in this distributed system. This task is simple if all nodes execute any protocol correctly. Interesting cases do however occur if we consider different types of node failures. We will assume that a node can *crash*, i.e., stop participating in the protocol at any point in time. Alternatively, nodes can show *Byzantine* behavior, i.e., they can behave arbitrarily, can choose to send different messages to different nodes or not to send any message at all. The Byzantine nodes are assumed to be controlled by an omnipotent adversary. The adversary knows the protocol and has knowledge about all message contents sent over authenticated channels. Moreover, the Byzantine adversary can also control message scheduling. The only restriction that we require from the Byzantine adversary is that it can decide which nodes it will control at the beginning of the algorithm and is not allowed to change this decision later on. Nodes that neither crash nor are Byzantine will be called *correct*.

We will differentiate between two communication models that determine how well the system is synchronized. We therefore make use of the definitions of synchronous and asynchronous communication for the message passing model as follows:

Synchronous communication: The communication is divided into discrete rounds. In a round, each node can send a value, receive values of all other correct nodes, and perform a local computation. All messages sent by a correct node in a round will be received by its recipients in the same round.

Asynchronous communication: There is no upper bound on the delay between sending and receiving a message. Every message sent by a correct node will eventually arrive at the destination.

In the shared memory model, we will consider a setting where each node is associated with a register in the memory. It can perform two operations: it can either write a value to its own register in the memory or retrieve the information from all registers by reading the memory. It is therefore reasonable to assume that the messages that are written to the memory instantly become available for other nodes to read. The nodes themselves do not have to be synchronized, however, and can perform their operations at different points in time. We therefore make use of the definitions similar to Dolev et al. [47] and differentiate between synchronous and asynchronous nodes:

Synchronous nodes: There exists a constant $\Delta > 0$ such that any interval between two operations executed locally by a single node is bounded from above by Δ. The upper bound Δ is known to all nodes.

Asynchronous nodes: The time between two operations of a node is not bounded. However, in an infinite protocol run, each correct node must perform infinitely many operations. Otherwise, the node is called faulty.

The Byzantine agreement protocol must generally satisfy the following standard conditions:

Agreement: All correct nodes agree on the same value upon termination.

Termination: All correct nodes must terminate after executing a finite number of operations.

All-Same Validity: If all correct nodes have the same input value b, they must agree on b at the end of the protocol.

Correct-Input Validity: The nodes agree on the value that at least one of the correct nodes has proposed.

The All-Same validity condition is the weakest validity condition possible, as it only prevents a system from agreeing on a predefined value. In this dissertation, we will therefore sometimes refer to the All-Same validity condition simply as the validity condition. Observe further that the Correct-Input validity is equivalent to the All-Same validity if the input values of nodes are binary. However, in the multi-valued setting where all nodes have different input values, the Correct-Input validity cannot be satisfied. This is because any correct node is not differentiable from a Byzantine node which follows the protocol using its own input value. We therefore introduce a relaxed variant of the Correct-Input validity:

Any-Input Validity: The nodes agree on the value that at least one of the nodes has proposed. This value is not required to be proposed by a correct node.

Fisher, Lynch and Paterson [53] showed that it is impossible to achieve consensus deterministically in an asynchronous communication model. The presented consensus properties can therefore be weakened such that each of the properties is only satisfied with high probability (w.h.p.). We call the relaxed properties *weak agreement, weak termination* and *weak validity* (for weak All-Same validity) respectively.

In the following sections, we will present known results on the message passing and the shared memory model that are used throughout the dissertation.

2.1.1 Synchronous BA in the Message Passing Model

Synchronous Byzantine agreement in the message-passing model has been widely explored in distributed computing. The quality of algorithms in this model has been estimated by considering the number of synchronous rounds - the *time complexity* - or the total size of messages exchanged during the protocol - the *message complexity* - for a worst-case run of the algorithm. In [72, 100], the authors showed that three nodes cannot establish agreement in the presence of one Byzantine node, even if the communication system is synchronous. Given n nodes, it was shown for the synchronous model that at least $t + 1$ rounds are required to establish agreement [52], where $t < n/3$ is the number of Byzantine nodes in the system. Berman et al. later proposed the Phase Queen [24] and the Phase King Algorithms [25] that both match this lower bound for binary input values.

The ideas from the Phase King Algorithm will be used several times throughout this dissertation. It is based on the fact that $t + 1$ nodes can be chosen as "leaders" in the algorithm, each leader associated with one phase of the algorithm. The algorithm can use multivalued inputs and is only required to satisfy the All-Same validity condition. Each phase of the algorithm is divided into three communication rounds: in the first round, the nodes share their input values and check whether All-Same validity holds; in the second round, nodes that satisfy All-Same validity broadcast this knowledge, and other nodes adjust their values if necessary; in the last round only the leader that is called "king" in the protocol is allowed to communicate, this leader simply shares its own value. The idea is that after a round with a correct node as a king, all nodes will have the same output value. In the case of a Byzantine king, the All-Same validity remains preserved, but the correct nodes might have different values at the end of the round if their input values were different. Algorithm 2.1 presents this protocol in pseudocode.

Besides protocols based on leaders, where one node dictates the output of the algorithm, there are also versions where all nodes participate by sending redundant information. In particular, the nodes would exchange their input values in a first round of the algorithm, forward all received values in the next round, and forward all information they have received so far in every following round. Such an algorithm corresponds to the so-called Interactive Consistency (IC) problem, where the nodes can agree on all input values of all nodes in the system at the end of the algorithm. Algorithms

Algorithm 2.1 Phase King Protocol for $t < n/3$ (code for node i)

```
1:  for phase i = 1 to t + 1 do
        Communication Round:
2:      Broadcast input value v_i
3:      receive guesses v_j from all other nodes
4:      if some value v is received ≥ n − t times then
5:          Broadcast("propose v")
6:      end if
7:      if some "propose v" received > t times then
8:          Adopt input value v_i := v
9:      end if
        King Round (only the King node executes this round):
10:     kingValue = v_king
11:     Broadcast("suggest kingValue")
        Decision Round:
12:     if "propose v" received < n − t times in Line 5 then
13:         v_i = kingValue
14:     end if
15: end for
```

for interactive consistency have been considered in [52, 100]. While such algorithms can establish agreement within $t+1$ communication rounds, the size of the broadcast messages increases by a factor of n in every round. This strategy leads to exponential message complexity. While repeating the forwarding step for $t + 1$ rounds is costly, already forwarding information for one round can improve the quality of the output, as will be shown in Chapter 3. The forwarding step is the so-called reliable broadcast that was introduced in [29, 112]. Algorithm 2.2 presents a reliable broadcast protocol for the synchronous communication model in pseudocode. In the first round of this protocol, one node – the sender – broadcasts a binary value. All nodes that have received a value in the first round broadcast an echo message for this value in the second round. If some node did not receive a value, but sufficiently many echoes for this value, it broadcasts an echo for the forwarded message in the third round. Note that the nodes might echo two different values this way, because a Byzantine sender could send different values to different nodes. Finally, if a node receives sufficiently many echoes for a value in both rounds, it will accept this value. If there are $n/4 < t < n/3$ Byzantine nodes participating in the protocol, the correct nodes might accept more than one value from the sender. For $t < n/4$ Byzantine nodes, at most one value from the sender will be accepted. The property of ac-

Algorithm 2.2 Synchronous Reliable Broadcast (code for node u)

```
1: Broadcast own input bit msg(u)
2: for all received msg(v) do
3:     Broadcast echo(u, msg(v))
4: end for
5: for all echo(w, msg(v)) received from at least n − 2t nodes w do
6:     if not echoed bit msg(v) before then
7:         Broadcast echo(u, msg(v))
8:     end if
9: end for
10: for all echo(w, msg(v)) received from at least n − t nodes w do
11:     Accept bit msg(v)
12: end for
```

cepting at most one value per sender is desirable in the Byzantine setting, because one would not want to accept too many Byzantine values in total. Throughout this dissertation, we will therefore assume that reliable broadcast is applied with $t < n/4$ Byzantine nodes.

Reliable broadcast guarantees the following properties for every broadcast input value:

- If a correct node has not broadcast a message, this message will not be accepted.
- All correct messages will be accepted within one reliable broadcast round.

In the synchronous communication model, Lines 5 to 12 can be repeated several times. In this case, the protocol would also satisfy the following property:

- If one correct node accepts a message reliably, all other correct nodes will accept the same message by the end of the following round.

Observe that the above properties can also be satisfied by slightly simplified protocols in the synchronous model. The reason for presenting this version is that it can be directly used in the asynchronous communication model. The only required adjustment is to let the nodes execute an echo operation upon receiving $n - 2t$ and $n - t$ messages for each value respectively instead of relying on synchronous rounds. Repeating Lines 5 to 12 is not necessary in the asynchronous setting.

2.1.2 Asynchronous BA in the Message Passing Model

For the asynchronous model, the FLP impossibility result [53] states that there is no deterministic agreement protocol that can tolerate even one Byzantine failure. The first solution to the asynchronous Byzantine agreement problem was proposed by Ben-Or [23], who used a trivial shared coin. This algorithm can tolerate up to $t < n/5$ Byzantine nodes and has constant expected running time for $t \in O(\sqrt{n})$. Mostefaoui et al. [92] improved the resilience of Ben-Or's algorithm to satisfy the optimal bound of $t < n/3$ Byzantine failures by adjusting the reliable broadcast to echo values instead of messages. The three papers by King and Saia [67–69] claim to have improved the Byzantine shared coin to run in expected polynomial time using spectral methods.

In Chapter 6, we will focus on the Ben-Or framework for solving asynchronous Byzantine agreement and the corresponding concept of a shared coin. The Ben-Or framework is often also referred to as a voting framework and it is a two-step protocol: in the first step, the nodes check whether the All-Same validity is satisfied and if true, the corresponding nodes terminate. In the second step, a new input value is chosen for the next communication round. This step is divided into two parts - the first part is a voting phase where the nodes choose their new input value deterministically if they suspect some other nodes to have terminated due to the validity condition; the second part is a randomized decision, where undecided nodes choose their new input value according to a global coin. An efficient implementation of this global coin is indispensable for voting protocols: if there is an efficient global coin, asynchronous Byzantine agreement can be solved efficiently using the Ben-Or framework. Algorithm 2.3 presents a possible implementation of the Ben-Or framework.

As in the synchronous communication model, the quality of an algorithm can be measured by either considering the message complexity or the round complexity. While the former definition of message complexity directly applies in the asynchronous model, the definition of round complexity has to be adjusted. A round can also be defined in an asynchronous model, see e.g. [29]: in a round, the nodes broadcast a message, wait for sufficiently many messages from other nodes, and perform some local computation. As some nodes might be delayed when sending and receiving messages, the rounds are defined locally with respect to the individual nodes. The Ben-Or protocol that was proposed in [23] has an expected exponential round complexity. This is because the predefined coin used in Step 11 of Algorithm 2.3 is a fair coin that each node flips locally and independently of all other nodes. In cases when All-same validity is not satisfied, the algorithm only terminates if at least $\lfloor n/2 \rfloor + 3f + 1$ nodes have the same coin outcome.

Algorithm 2.3 Ben-Or Framework for $t < n/10$ (code for node u)

1: $x_u \in \{0, 1\}$ ▷ input bit
2: r=1 ▷ round
3: Broadcast **propose**(x_u,r)
4: **repeat**
5: Wait until $n - f$ **propose** messages of current round r arrived
6: **if** at least $\lfloor n/2 \rfloor + 3f + 1$ **propose** messages contain same value x **then**
7: $x_u = x$, decided = true
8: **else if** at least $\lfloor n/2 \rfloor + f + 1$ **propose** messages contain same value x **then**
9: $x_u = x$
10: **else**
11: choose x_u randomly, according to a predefined coin
12: **end if**
13: r = r+1
14: Broadcast **propose**(x_u,r)
15: **until** decided (see Step 7)
16: decision = x_u

In the literature, a global coin is either generated in a distributed way, as in [23], or is provided to the nodes by a trusted third-party. We will discuss this trusted third-party strategy next. A global coin can be seen as an oracle which is requested every time a random decision is required by the algorithm. This global coin is often given to the nodes as an encrypted bitstring containing $\Omega(n)$ random coinflips at the beginning of the protocol. As we will show next, the encryption of such a bitstring is important, since Byzantine nodes can otherwise prevent the Ben-Or framework from terminating.

Lemma 2.4. *Let the global coin be defined by a random bitstring which is revealed to the nodes at the beginning of the Ben-Or framework. Then, there is a Byzantine strategy that can prevent the Ben-Or framework from terminating.*

Proof. Let -1 and $+1$ be the binary input values of the Ben-Or framework. As an example, we can let the nodes terminate in the first step of the Ben-Or framework if they see a majority of $n-2t$ for one of the binary input values, and deterministically vote for a value if they see an $n-3t$ majority for it. In all other cases, the new input value is chosen according to the bitstring value of the current round. The following argument can be adjusted to work for

different thresholds as well: Assume that the algorithm starts with an input where $n-3t$ nodes have input value $+1$, and $3t$ other nodes have input value -1. Further assume that the first bit of the bitstring is -1, and that the second random bit of the bitstring is known to the Byzantine scheduler. If the second random bit in the bitstring also is -1, then a Byzantine scheduler can let $n-3t$ nodes see $n-3t$ messages of $+1$, and therefore deterministically choose the value $+1$ in the voting step. The remaining $3t$ correct nodes will receive strictly less than $n-3t$ values of $+1$ and therefore have to rely on the value of the shared coin, which is -1 in this round. This way we have created the same input for the next round. If the second bit is $+1$, then a Byzantine scheduler can make $3t$ correct nodes receive $n-3t$ messages of $+1$ and thus choose the value $+1$ deterministically. The remaining correct nodes will receive strictly less than $n-3t$ messages of $+1$ and will rely on the value of the shared coin, thus choosing -1 as the new input value. The Byzantine scheduler has thus generated a situation where the input is opposite to the input of the previous round. □

It remains an open question of whether asynchronous Byzantine agreement can be solved efficiently in the message passing model without relying on cryptographic assumptions. If cryptographic assumptions are used, Byzantine agreement can be solved in expected constant number of rounds. The first such implementation is due to Rabin [105] and it uses Shamir's threshold secret sharing. Rabin's algorithm relies on the fact that a dealer provides the random bitstring. Chor et al. [39] proposed the first algorithm where the nodes use verifiable secret sharing in order to generate random bits. Later work has focused on improving these algorithms in terms of resilience [35] and practicability [34].

In our model, we make no cryptographic assumptions and assume no trusted oracles. In this dissertation, we therefore consider the second case where the global coin is generated by the nodes in a distributed way. Due to Lemma 2.4, our model cannot tolerate a bitstring, and therefore the nodes need to generate a separate random value for each Ben-Or round. The question of generating an efficient global coin therefore becomes a question of implementing an efficient shared coin that can tolerate Byzantine values. All existing shared coins in the public channel model have expected exponential running time when Byzantine nodes are present in the system. We are therefore in search or an efficient shared coin, i.e., a shared coin that can generate each of the two outputs with at least polynomial probability:

Definition 2.5 (Polynomial-Time Shared Coin)**.** *Let L be the set of possible local views of the nodes. A polynomial-time shared coin is a function $s : L \mapsto$*

$\{-1, +1\}$, *where*

$$\Pr[\forall i : s(l_i) = -1] \geq \frac{1}{d \cdot n^c} \quad \textit{and} \quad \Pr[\forall i : s(l_i) = +1] \geq \frac{1}{d \cdot n^c}$$

for two constants $d, c > 0$, *where* $l_i \in L, i \in [n]$, *are the local views of the same round.*

In the message passing model, the shared coin is usually implemented using reliable broadcast. Reliable broadcast was first proposed by Srikanth and Toueg [112] as a method to simulate authenticated broadcast. There is also another implementation which was proposed by Bracha [29]. Today, a lot of variants of reliable broadcast exist, including for example the FIFO broadcast [2]. A good overview of the broadcast routines is given by Cachin et al. [33]. In order to achieve an efficient shared coin, King and Saia [69] develop a stronger reliable broadcast routine called the blackboard model in order to communicate large random values reliably. This model will be discussed in Chapter 6.

2.1.3 BA in the Shared Memory Model

Other than in the message passing model, nodes in the shared memory model do not communicate with each other, but instead only with the shared memory. It can therefore be assumed that each node has a register in the memory where it can write its values and that it has a read access to all registers of the memory. This way, every value that was written to the memory can be observed by all nodes that have access to it. The shared memory model is not stronger than the message passing model, as the nodes might overwrite their own values before other nodes in the system have observed them. Alternatively, the nodes might only share one register and all have write access to this register. In such a model, nodes might overwrite each others' values and thus make agreement hard.

The problem of sharing data among several processors in a system has been extensively studied in the literature. Early solutions to this problem required mutual exclusion [41, 45], i.e., only one process was allowed to access and alter the memory at a time while the other processors were denied access. The first discussion on the wait-free implementation of shared objects goes back to Herlihy [58]. In this paper, he defined the *consensus number* as the maximum number of nodes that can establish consensus in the system using arbitrarily many shared objects. According to this definition, a hierarchy of shared objects can be established. In particular, Herlihy showed that the consensus number of read-write registers is 1, i.e., consensus cannot be established by two processors using read-write registers.

In Chapter 5, we will consider a shared memory model having n read and write registers, where each node has write access to only one of the n registers. If only crash failures are considered in such a system, there exists a simple implementation of the shared coin from Definition 2.5. Algorithm 2.6 presents an implementation of such a consensus protocol that can tolerate up to $t < n/2$ crash failures.

Algorithm 2.6 Shared Coin Tolerating Crash Failures (for node u)

```
1: n_u = 0
2: c_u = 0
3: while true do
4:     Choose new local coin c = +1 with probability 1/2, else c = −1
5:     Write c_u = c_u + c and n_u = n_u + 1 to u's register in the shared
   memory
6:     Set C = Σ_u c_u
7:     if Σ_u n_u ≥ n^2 then
8:         return sign(C)
9:     end if
10: end while
```

The first polynomial algorithm for the shared memory model that uses a shared coin was proposed by Aspnes and Herlihy [11] and required exchanging $O(n^4)$ messages in total. Algorithm 2.6 requires exchanging $O(n^3)$ messages and it is a variant is due to Saks, Shavit, and Woll [108]. Bracha and Rachman [30] later reduced the number of messages exchanged to $O(n^2 \log n)$. The tight lower bound of $\Omega(n^2)$ on the number of coinflips was proposed by Attiya and Censor [13] and improved the first non-trivial lower bound of $\Omega(n^2/\log^2 n)$ by Aspnes [10].

It has been shown that it is possible to simulate the read and write commands from the shared memory in the message passing model. This is done by the so-called ABD simulation [12]. A write operation can for example be simulated by a reliable broadcast round, where a node reliably broadcasts a value and considers it written to the memory once it accepts its own value. In a read operation, a node can request all accepted values from other nodes and restore a total view. If the ABD simulation is applied to Algorithm 2.6, one achieves a consensus algorithm with crash failures in the message passing model with a message complexity of $O(n^3)$. Alistarh et al. [5] improved the number of exchanged messages to $O(n^2 \log^2 n)$ using a binary tree that restricts the number of communicating nodes according to the depth of the tree.

Byzantine agreement in the shared memory was first considered by Malkhi et al. [79]. They used the concept of sticky bits [102] and access control lists in order to restrict Byzantine power. Sticky bits are bits that remain in the memory and cannot be overwritten. They also showed that Byzantine agreement is impossible in their model if $t > n/3$ and provided a protocol that could tolerate $(\sqrt{n}-1)/2$ Byzantine failures. Alon et al. [7] later showed that the bound on the resilience is tight by using exponentially many sticky bits.

2.2 Chapter Overview and Related Work

Chapter 3 - Byzantine Agreement with Interval Validity

Overview Byzantine agreement is well studied in the binary setting where each node has the input value 0 or 1. Many applications must however be able to handle real numbers $\mathbb{R}$ or natural numbers $\mathbb{N}$. This setting is referred to as *multivalued* Byzantine agreement [119]. Chapter 3 focuses on synchronous Byzantine agreement, as defined in Section 2.1.1, on input values that can be ordered. The idea is to establish agreement on a value that was an input value of some correct node. Algorithms that solve this problem have been proposed in [54,117,119]. These algorithms assume that the majority of the nodes have the same input values. If there is no clear majority, some leader node might decide on a value that all nodes will adopt, or the nodes choose a preselected value. These algorithms need $t+1$ rounds to establish agreement, which is optimal in the synchronous model. The algorithms will however agree on an arbitrary value if there is no majority among the input values.

In Chapter 3, we present a protocol that establishes agreement on a value that is an approximation to the k^{th} smallest of all correct input values. Given totally orderable inputs, we are looking for an output close to the k^{th} largest or smallest value, and depending on the number of Byzantine nodes we can tolerate a solution in an *interval* around this value. We will refer to this condition as *interval validity*. In Section 3.3, we present an algorithm that will achieve consensus on a value that satisfies interval validity. Our algorithm is optimal in that it tolerates the maximum possible number of $t < n/3$ Byzantine nodes. Our algorithm is also optimal in how close the chosen value is to the k^{th} value thanks to a matching lower bound.

Related Work Similar approaches have been considered for the special case of the median: Doerr et al. [46] consider the Power of Two Choices to establish agreement on a value which is close to the median in the asynchronous message passing model. In their protocol, each node requests the

values of two nodes chosen uniformly at random among all nodes and updates its value to the median of the two requested values and its own. The authors showed that for $t \in O(\sqrt{n})$ the system stabilizes with a consensus value that is between the $(n/2-c\sqrt{n \log n})$-largest and the $(n/2+c\sqrt{n \log n})$-largest value. The same problem was considered for the synchronous message passing model by Stolz et al. [114]. In this paper, the authors proposed an algorithm that computes an approximation of the median within $t + 1$ rounds. Their approximation is not optimal compared to the bounds of any deterministic algorithm for this model. We will use these ideas to derive an approximation to the k^{th} smallest value and show that our method can be adjusted to solve the median problem optimally.

Dolev et al. [48] proposed a deterministic algorithm for *approximate* Byzantine agreement in the asynchronous message passing model. In this relaxed setting, nodes do not establish consensus on an exact value, since this is impossible [53]. They instead converge towards a consensus value in every round. In the proposed algorithm, the values of all nodes converge towards the mean of the correct values. Fekete [51] later improved the running time of the algorithm and proposed an algorithm for approximation that achieves exact consensus in the synchronous model, if it iterates for $t + 1$ rounds. Approximate agreement can guarantee that the values of correct nodes will be inside an arbitrarily small interval after sufficiently many iterations but cannot solve the exact Byzantine agreement problem unless the lower bounds for exact consensus are satisfied. Another relaxation of the Byzantine agreement problem is k-set agreement, where the nodes try to agree on values that are within some common set of size at most k [36, 86, 103].

In the spirit of [87, 88, 120], we will present how our method can be applied to Byzantine vector consensus. This is a generalization of multivalued agreement that allows multidimensional input values. Previous work mostly concentrated on finding a value that is within the convex hull of all correct values. While this method is efficient for approximate agreement in the asynchronous message passing model, the exact agreement in the synchronous message passing model requires an exponential number of computations to determine the convex hull in the presence of Byzantine nodes. Xiang and Vaidya [124, 125] introduced two relaxations of the convex hull - the k-relaxed and the (δ, p)-relaxed Byzantine vector consensus. The former requires the consensus value to be inside the projections of the convex hull onto any k dimensions of the vectors and the latter requires the value to be within distance δ to the convex hull. They show that their relaxation cannot be used to improve the number of Byzantine nodes that can be tolerated by the system. We will relax the validity condition from the convex hull to a box and apply our proposed k^{th} smallest value algo-

rithm. This adjustment allows us to achieve exact consensus in $O(d(t+1))$ rounds, where d represents the dimension. It can furthermore tolerate up to $\lceil n/3 \rceil - 1$ Byzantine nodes.

Chapter 4 - Byzantine Preferential Voting

Overview This chapter introduces a version of multi-dimensional Byzantine agreement, where the input values of the nodes are preference rankings of three or more candidates. We show that consensus on preferences, which is an important question in social choice theory, complements already known results from Byzantine agreement. In addition, preferential voting raises new questions about how to approximate consensus vectors. We propose a deterministic algorithm to solve Byzantine agreement on rankings under a generalized validity condition, which we call Pareto validity. Pareto validity states that, if all correct nodes prefer one candidate over the other, then the consensus ranking should rank these two candidates in the same way. Note that this validity condition nicely generalized the All-Same validity condition proposed in Section 2.1. The results for Pareto validity are then extended by considering a special voting rule which chooses the Kemeny median as the consensus vector. The Kemeny rule was first proposed in [62,63]. The corresponding Kemeny median satisfies many desirable properties for a consensus ranking which will be presented in Section 4.1. Note that this voting rule was shown to be NP-hard to compute for an increasing number of candidates and already for four voters in [18, 50]. At least three different 2-approximation algorithms for the Kemeny median have been proposed in [4] and [44]. In [4], the approximation ratio was improved to 4/3 using randomization and later derandomized in [121]. A good overview over the Kemeny rule and an extended introduction into social choice theory can be found in [32]. In Section 4.3, we derive a lower bound on the approximation ratio of the Kemeny median that can be guaranteed by any deterministic algorithm. We then provide an algorithm that approximates the solution of the Kemeny rule in the presence of Byzantine voters and prove that this algorithm computes the best possible approximation. To our knowledge, this is the first non-trivial generalization of multi-valued Byzantine agreement to multiple dimensions which can tolerate a constant fraction of Byzantine nodes.

Related Work Byzantine agreement with more than two input values has mostly been considered in approximate agreement [48, 51], where the input values of the nodes converge towards some value over rounds. More recent results seek to establish agreement on a value that makes sense for applications. In [46], the values converge towards a value at most $\sqrt{n \log n}$

positions away from the median. In [84,114] an exact algorithm to establish agreement on a value that is at most $t/2$ positions away from the median or t positions away from a minimum or a maximum was proposed. In [87,88,120], Byzantine agreement was further generalized to several dimensions. There, the nodes converge to a vector inside the convex hull of all correct input vectors. In [37,118] the authors consider voting in Byzantine systems, they do however only focus on single winners that are determined by applying the plurality rule to the top alternatives of the rankings, a setting which corresponds to standard Byzantine agreement. All previous approaches for multiple dimensions struggle to derive an algorithm which either can tolerate a constant fraction of Byzantine nodes independent on the number of dimensions, or find a solution that is not trivial.

In social choice theory, Byzantine behavior can be interpreted as manipulation of a ballot in an election, in which the manipulating party has full knowledge about all votes. Bartholdi et al. [19] defined manipulation as a preference profile where one single voter can change its ranking such that this voter's most preferred candidate wins the election. Groups of voters have also been considered in this context, but mostly from the perspective of how hard it is for a group of nodes to manipulate the voting result given a certain voting rule [26,42]. Other types of Byzantine behavior have been considered with respect to the robustness of proposed voting rules. In [20], the authors investigate the robustness of Borda's mean and median in the presence of outlier ballots. In [104], the robustness of scoring rules is considered under arbitrary noise which is described in terms of pairwise swaps of candidates in the ranking of one voter.

Chapter 5 - The Append Memory Model

Overview This chapter presents a novel shared memory model that simplifies the analysis of consensus on a Chain and a DAG. In this new model, referred to as the append memory model, nodes are allowed to write new values to the unordered memory, but not to overwrite already existing values. We show that although this model differs from the standard shared memory model with n shared read-write registers that was introduced in Section 2.1.3, many known results from the shared memory model still hold in the append memory model. In Section 5.2, we therefore show that asynchronous consensus cannot be solved in this append memory model, as the nodes cannot uniquely define the ordering of concurrently appended commands. In Section 5.5, it is shown that this result also holds for the asynchronous communication model with randomized memory access. We further show that our proposed model is not stronger than the message passing model, as it can be simulated in the message passing at a high message com-

plexity cost (see Section 5.4). The advantage of the append memory model is that it simplifies the analysis of Blockchain protocols. In Section 5.5, we will therefore compare the analysis of the DAG and the Chain in the append memory. We will show that Byzantine agreement on the DAG can achieve almost optimal resilience of $< 1/2$, while Byzantine agreement on the Chain highly depends on the append rate of the correct nodes. Our results suggest that the DAG is not only a better model for Byzantine agreement because of its simplicity compared to the Chain, but also because it achieves an optimal resilience.

The analysis in Chapter 5 deviates from the previous work in several ways. Many papers try to directly solve Nakamoto consensus in the message passing model, thereby oversimplifying the communication model [55] or falsely calling a synchronous communication system (partially) asynchronous [66, 98]. Instead, we focus on deriving a shared memory model for Blockchain protocols, which allows us to assimilate the local views of the nodes and thereby derive simpler protocols for Blockchain and DAG. Note that the append memory is not as strong as the concept of sticky bits [79] since it does not make use of registers that implicitly solve consensus for two parallel writes.

Related Work The recent wave of interest in blockchain systems was sparked by Satoshi Nakamoto's Bitcoin protocol [93] – a distributed cryptocurrency system based on peer-to-peer communication and the construction of a Blockchain. The Nakamoto consensus needs to satisfy two main properties: consistency and liveness. The first rigorous analysis of Nakamoto consensus on a Blockchain was given by Garay et al. [55]. In their work, they analyzed the blockchain in the synchronous communication network, assuming that all messages of the current round arrive at the beginning of the next round. They showed that Nakamoto consensus solves Byzantine agreement with validity under the assumption that the Byzantine nodes have strictly less than 1/3 of the hashing power of the network. They further propose a more elaborate consensus protocol on the Blockchain for which they show a resilience of up to 1/2. Many following attempts were made in order to formalize Nakamoto consensus in a more general model: Pass et al. [98] extend the synchronous model of [55] and consider a δ-synchronous network, where δ is an upper bound on message delay known to all nodes in the network. Pass and Shi [99] later simplified the previous model mostly for didactic purposes. Another formalization of consistency of Nakamoto consensus was given by Kiffer et al. [66], who use a Markov chain-based analysis to prove consistency in a synchronous setting.

All aforementioned papers note that the analysis of Nakamoto consensus generally is involved, and therefore needs complicated models in order to describe their system rigorously. The first simple analysis of Nakamoto consensus was provided by Ling Ren [106]. Instead of focusing on the definition of the communication model, [106] focuses on a correct analysis of the chain growth and therefore differentiates between blocks which are "non-tailgaters" and "loners". The former describes blocks which were mined after seeing the last correct block in the system, while the latter denotes blocks which are non-tailgaters and are not non-tailgated. This way, both, forks in the Blockchain introduced by the Byzantine nodes, as well as forks produced by the correct nodes, are taken care of. Ren further shows that Nakamoto consensus satisfies consistency and liveness under the honest majority assumption, provided that the block generation rate of the correct nodes is much larger than the communication delay.

Other structures for reaching Nakamoto consensus have also been considered in the literature. In [111], it was shown that the so-called inclusive Blockchain, which relies on the DAG structure, can provide safety in the Blockchain protocol even if the system is asynchronous for a short period of time. The DAG structure is usually considered under one of the tie-breaking rules, such as the GHOST protocol [111] or the pivot chain [74] rule.

Observe that, while many protocols also considered Byzantine agreement besides Nakamoto consensus, consistency and liveness actually do not necessarily require consensus as a building block. This was first shown by Gupta [57]. In follow-up work, Guerraoui et al. [56] show that Nakamoto consensus has a consensus number 1. Other than the protocols mentioned previously, such systems work in the fully asynchronous setting but do not satisfy consistency at any point of time in the protocol, and would therefore require checkpointing techniques in order to be applied in cryptocurrency systems.

Chapter 6 - Asynchronous Byzantine Agreement and Deep Reinforcement Learning

Overview In this chapter, we consider the asynchronous Byzantine agreement problem based on the Ben-Or framework that was introduced in Section 2.1.2. The chapter is inspired by some recent papers that challenge the unwritten law by trying to expose and expel Byzantine nodes. This strategy is usually applied in protocols with computationally bounded adversaries, where the nodes are asked to sign their messages and the Byzantine behavior can be exposed by a single node who forwards Byzantine messaged to all nodes. In the case of a computationally unbounded adversary, a single node cannot expose Byzantine behavior due to the "word against word" sit-

uations. The basic idea to overcome this problem is to let all nodes produce many random bits. Because of the central limit theorem, the sum of these random bits will deviate from the expectation, and this deviation can be used for a decision. Byzantine nodes can in that case for example try to prevent the decision and in the process probably exhibit "unlikely randomness". Based on this *evidence* they can then be exposed and expelled.

We first focus on an efficient shared coin implementation for this framework, by relying on the blackboard model from [69]. Using the blackboard model, we define a reliable broadcast routine which implies strong similarity properties on the local views of the nodes. We show that this model can be directly used to generate a shared coin that has expected polynomial running time in the crash failure model. Moreover, we discuss the possibility to rely on the so-called blackboard broadcast in order to solve asynchronous Byzantine agreement. Our results suggest that the algorithms proposed in [67–70] can be used to generate an efficient shared coin in the crash failure setting, but the arguments are not sufficient for the Byzantine setting. We furthermore discuss the role of Byzantine values and Byzantine scheduling strategies as necessary conditions to prevent asynchronous Byzantine agreement from terminating within expected polynomial running time.

We then investigate the possibility of using Deep Reinforcement Learning (DRL) in order to find possible new Byzantine strategies and develop more robust algorithms. In particular, we show that DRL can be used to simulate Byzantine behavior that is needed in order to make the standard versions of the Ben-Or algorithm inefficient. We observe that the behavior that a trained Byzantine agent learns corresponds to the worst-case strategies that have been discussed in Section 2.1.2. We then discuss the possibility of using self-play in order to not only train the Byzantine agent, but also the algorithm. These preliminary results show that DRL is indeed a good tool for simulating Byzantine behavior and that it can help understand future Byzantine agreement algorithms better.

Related Work Deep Reinforcement Learning has recently been applied to various graph network problems, for example, in [6, 94], and has shown promising results. In [94], Nakashima et al. used DRL with Graph Convolutional Networks for channel allocation in WLANs. In [6], Almasan et al. apply Graph Neural Networks in combination with DRL to a routing optimization problem. In this dissertation, we will use the Deep Q-Network (DQN) algorithm that is based on Q-learning. Q-learning was first introduced by Watkins [122]. It has been developed since and has found its application in fault-tolerant control in the work of Hua et al. [60]. Later it

was also applied to fault handling in self-organizing system by Mismar et al. [90].

In this dissertation, we are interested in a simulation of Byzantine nodes using DRL. An example of how DRL can be used for simulations is given by Lee et al. [73], who simulated crowd navigation. Hou et al. [59] later use DQN to simulate worst-case adversarial behavior in blockchain incentive mechanisms, which is closely related to our goals. In their paper, the authors use DRL to identify attack strategies on incentive protocols beyond selfish mining and show that classical selfish mining attacks are not as effective when multiple attackers are present in the system.

Reinforcement learning has also been recently applied to various combinatorial optimizations problems, see for example [17, 22, 65, 71, 101, 128]. Such methods often assume a distribution of the inputs or learn a specific algorithm type and are thus not generalizable. In order to develop general strategies, self-play has been proposed in the literature. In self-play, an algorithm agent is competing against an adversary agent, who is looking for worst-case input values. Such approaches have been tested for many complex games, such as Go, Chess or AlphaZero, and could defeat human professional players [109, 110]. Bansal et al. [16] and Baker et al. [15] considered self-play in multi-agent environments where the agents compete against each other. The former paper considers a soccer goal shooting task, while the latter focuses on hide-and-seek games. Both papers show that their agents can produce behaviors that are more complicated than their environments. Besides computer games, self-play has also been applied to theoretical problems, such as combinatorial optimization problems [126] or graph coloring [61].

Byzantine faults are also of interest in distributed machine learning, where the goal is to split computations among several machines for scalability reasons and let the machines collaborate in order to learn a common model [1, 28, 43, 75, 76]. The first papers that consider Byzantine failures [27, 38] focus on the stochastic gradient descent (SGD). Blanchard et al. [27] propose Krum - an aggregation rule that satisfies certain resilience properties - and show that SGD converges under this rule when $2t + 2 < n$. Chen et al. [38] show that choosing the geometric median of means of the gradients is sufficient to make the SGD converge for $2(1 + \varepsilon)t \leq n$. Some follow-up results to their work consider the non-convex loss functions [89] or the possibility to avoid saddle-points [127].

3

Byzantine Agreement with Interval Validity

There exist several situations where nodes of a distributed system do not propose the same state. For example, all nodes of a distributed stock market system may have seen a different transaction first, and therefore propose their own transaction as the next one to be included in the common ledger. Other situations could occur when all nodes of a distributed auction system offer a slightly different price, or the nodes of a distributed flight control system are equipped with a height sensor and all sensors report slightly different altitudes, e.g., represented as floating point numbers. With Byzantine nodes participating in the decision, it would be not advisable to simply agree on any value.

A majority decision will also not help in systems where each node might propose a different outcome. Luckily, many distributed systems seem to have in common that the inputs of the Byzantine agreement algorithm can be ordered: some transactions have an earlier timestamp than others and altitude sensors will likely report at least slightly different heights. If the inputs are ordered, we can try to decide on a value that is not an outlier, as outliers coming from Byzantine nodes should be avoided. Instead, we want to decide on a value which makes sense: If we want our plane to operate

safely, we can think of avoiding outliers by choosing the median value. If we are interested in the price of honest bidders for our apartment, we are looking for a high bid, but not a goofy outlier.

The median is in some sense the safest value in a Byzantine setting, as it is robust against Byzantine attacks from both sides, i.e., it does not matter whether the Byzantine nodes propose high or low values. This chapter presents an algorithm that finds the optimal median in the Byzantine setting. The ability to choose the largest or smallest value in Byzantine environments also finds various applications. The need for a generalization to the k^{th} largest or smallest value is less obvious, but it is interesting in several cases as well. As an example, consider a distributed system with at most t Byzantine (arbitrarily malicious) nodes. In addition to these Byzantine nodes, it is assumed that there are nodes that are not Byzantine but also not correct. These nodes will generally follow the protocol, but they will not be completely honest about their input, e.g., agents who always bid a too high value. We do not want our result to be affected by these nodes. By going for the k^{th} smallest or largest value instead of the maximal/minimal/-median value, we can adapt nicely to such situations. In this chapter we will assume that the implementation is aware of such malfunctioning behavior of the system and chooses k accordingly before starting the algorithm.

3.1 Model

In this chapter, we consider Byzantine agreement in the synchronous message passing model, as it was introduced in Section 2.1.1. At the beginning of the computation, each node has an input value from a totally orderable domain, for example $\mathbb{R}$. The goal is to make all nodes agree on a common value that is close to the k^{th} smallest value of all correct input values by communicating in synchronous rounds. For simplicity, we assume that all input values are different. It is not necessarily a drawback if the Byzantine nodes propose values close to the k^{th} smallest value, while values that are far away should be omitted. In order to handle values that are too far away from the k^{th} smallest value, we introduce a new validity condition called *interval validity*. Let S be a sorted array containing the $n - t$ correct input values and refer to $S[k]$ as the k^{th} smallest value in this array. In each round, it is assumed that every node stores the values of all received messages in a sorted array R of size $n - t + f$. Note that $R[k]$ is not the same value as $S[k]$, since R also stores f Byzantine values. The validity condition for the k^{th} smallest value is defined as follows

Definition 3.1 (Interval Validity)**.** *Sort all input entries of correct nodes in an array S. A valid value is a value v that is close to the k^{th} smallest*

value of all correct nodes:

$$S\big[k - \lceil t/2 \rceil\big] \leq v \leq S\big[k + \lfloor t/2 \rfloor\big]$$

This validity condition does not guarantee that v is an input value of a correct node. We only require it to not be further away than $\lceil t/2 \rceil$ positions from the actual k^{th} value in S. For $t = 0$ the validity condition holds only for $S[k] = v$, which is the exact k^{th} smallest value. Note that this is not the same definition of interval validity as used in [21].

For brevity, we denote the subarray of S which starts with the a^{th} value and ends with the b^{th} $S[a, b]$. In contrast, $\big[S[a], S[b]\big]$ denotes the interval enclosed by the two values, i.e., correct and Byzantine values which lie inside the boundaries. The same notation is used for R, which also will be referred to as the local array of a node.

For the computation of the k^{th} smallest value, we use the geometric median. It is defined as the central value of an array of ordered numbers. For an even number of nodes, this value usually corresponds to the mean of the two central values. We adjust the definition of the median to agree on the smaller of the two central values in the case where a node receives an even number of values:

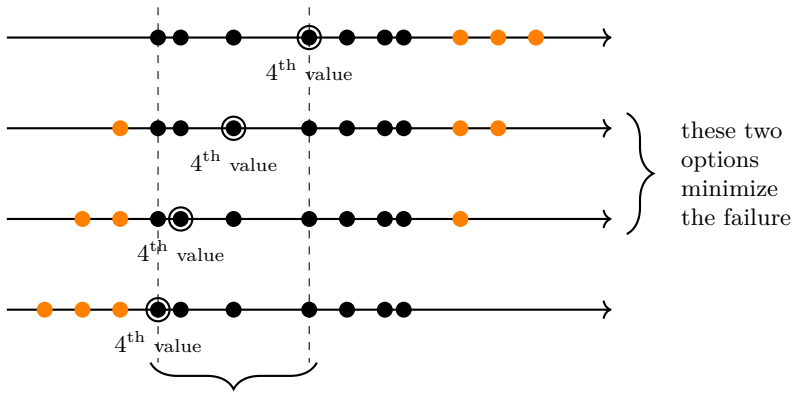
Definition 3.2 (Median)**.** *Given an array A of n values, the median is defined as $A\big[\lfloor n/2 \rfloor\big]$, i.e., the value at position $\lfloor n/2 \rfloor$ in the array A.*

This alternative definition enables the nodes to choose a valid median value.

3.2 Lower Bound for the Approximation of the k^{th} Smallest Value

In this section, we show that no deterministic algorithm can approximate the k^{th} smallest value better than by $\lceil t/2 \rceil$ positions in the presence of t Byzantine nodes. The quality of the approximation is calculated by the number of positions by which the approximate k^{th} smallest value is shifted from the actual k^{th} smallest value with respect to the array S. Two cases will be considered separately, the case where $k \in \big[\lceil t/2 \rceil + 1, n - \lfloor 3t/2 \rfloor\big]$ and the case where k is outside of these bounds. In the first case, the lower bound on the approximation value is $\lceil t/2 \rceil$. In the second case, we need to guarantee that the approximation value is inside the interval of all correct nodes. Otherwise, the Byzantine nodes might choose values that deviate arbitrarily from the values of the correct nodes. With this restriction, the approximation of the k^{th} smallest value can be up to t positions away from the actual value.

Figure 3.3: In this example we look for the 4^{th} smallest value in a system with $(n-t) = 7$ correct nodes and $t = 3$ Byzantine nodes. The correct nodes are shown in black, the Byzantine nodes in orange. All values are ordered according to the axis. On the first axis, the Byzantine nodes choose their value to be larger than all correct values. On the last axis, the Byzantine values are all smaller than the correct values. In between, the Byzantine nodes choose some values to be larger and some to be smaller. All four cases are not differentiable to the correct nodes since all nodes just see n values. The correct nodes do not know the position of the Byzantine nodes. Therefore, any of the circled correct nodes might be a candidate for the actual 4^{th} smallest value. Thus, the nodes must agree on a value that minimizes the distance to each of the four candidates.



Theorem 3.4. *Assume $k \in \left[\lceil t/2 \rceil + 1, n - \lfloor 3t/2 \rfloor\right]$. Then, no deterministic algorithm can choose a value that is closer than $\lceil t/2 \rceil$ to the actual k^{th} smallest value when t Byzantine nodes are present in the system.*

Proof. We consider $t + 1$ cases for which the values that the correct nodes received only differ in t values that were sent by the Byzantine nodes. We will show that the correct nodes are not able to distinguish the given views, while the k^{th} smallest values in any two views differ by up to t positions. Figure 3.3 shows such an example for $t = 4$ with t different values that might potentially be the k^{th} smallest value.

Let $\max(S)$ and $\min(S)$ respectively denote the maximum and minimum value of the correct nodes. We assume that a Byzantine node always sends

the same value to all other nodes, i.e., all nodes receive all t Byzantine values, and values that were sent by the same node are equal. In the first case, the t Byzantine nodes send values that are larger than $\max(S)$ to every correct node. In this case, the new k^{th} smallest value has the same position as before, i.e., the k^{th} smallest value is $R[k]$.

In the second case, we assume that one Byzantine node sends a value that is smaller than $\min(S)$ and the other $t-1$ Byzantine nodes send values that are larger than $\max(S)$. This way, the k^{th} smallest value is at position $k+1$ in the array R.

In the third case, we assume that two Byzantine values are smaller than $\min(S)$. This will shift the k^{th} smallest value by 2 positions in R etc.

In the last case, all t Byzantine nodes broadcast values that are smaller than $\min(S)$. Here, the Byzantine nodes shift the k^{th} smallest value to position $k+t$ in the new array, i.e., it is $R[k+t]$.

In any of the cases, a node knows that its array contains exactly t Byzantine values, but the cases are indistinguishable to the node. The k^{th} smallest value can therefore be any value from the subarray $R[k, k+t]$. Choosing a value closer to k would decrease the mistake in the first case and increase it in the last. A value closer to $k+t$ does the opposite. The value that minimizes the mistake is the median value which is at most $\lceil t/2 \rceil$ positions away from all solutions. Note that for odd values of t the value must be rounded up since the median of $R[k, k+t]$ lies between two values. It is thus not possible for a deterministic algorithm to be better than $\lceil t/2 \rceil$ positions away from the optimal solution. □

Theorem 3.5. *For k outside $\left[\lceil t/2 \rceil + 1, n - \lfloor 3t/2 \rfloor\right]$, any deterministic algorithm can be forced to choose a value further away than $\lceil t/2 \rceil$ but at most t positions away from the actual k^{th} smallest value.*

Proof. We consider the same $t+1$ cases as in the proof of Lemma 3.4. The median of the subarray $R[k, k+t]$ minimizes the mistake of approximating $S[k]$. If $k \leq \lceil t/2 \rceil$ the median of this subarray may be a smaller value than $S[1]$ which is the smallest correct value. The median does not satisfy the requirements for the approximation of the k^{th} value in this case since the values outside of $\left[S[1], S[n-t]\right]$ can be arbitrarily small or large. The closest value inside the interval of correct values is at position $t+1$. Therefore, the value at position $t+1$ is the one that minimizes the mistake to any $S[k]$ with $k \leq \lceil t/2 \rceil$. Analogously, the closest correct value for $k > n - \lfloor 3t/2 \rfloor$ is at position $n-t$. The guess of the k^{th} smallest value may deviate from the actual value by more than $\lceil t/2 \rceil$. If we are looking for the minimal or maximal values, i.e., $S[1]$ or $S[n-t]$, this value may deviate by t, since all

Byzantine nodes may choose values smaller than the smallest correct value or larger than the largest correct value. □

3.3 Algorithm for the k^{th} Smallest Value

In this section, we present an algorithm that selects an approximation of the k^{th} smallest value in the presence of Byzantine nodes. We will show that this algorithm gives the best approximation to $S[k]$ for all values of $k \in [0, n-t]$. As in the previous section, two cases are distinguished for which the bounds of the approximation differ: In the case where $k \in \left[\lceil t/2 \rceil + 1, n - \lfloor 3t/2 \rfloor\right]$, the algorithm finds a value that satisfies Definition 3.1. For k outside of $\left[\lceil t/2 \rceil + 1, n - \lfloor 3t/2 \rfloor\right]$ the solution can not be guaranteed to satisfy Definition 3.1. In this case, the solution will instead be at most t positions away from the actual k^{th} smallest value. We will also prove that our algorithm produces an optimal solution by showing that it matches the bounds from Section 3.2.

The main idea of the algorithm is to perform a step at the beginning where each node selects a new input value that is close to the actual k^{th} smallest value that we are looking for. Denoting this value the new input value of the node, we reduce the problem of establishing agreement with a special result to a multivalued agreement where nodes can agree on any value inside the interval of all correct values, i.e., inside $\left[S[1], S[n-t]\right]$. We use the ideas of the Phase King Algorithm proposed by Berman et al. [25] to establish agreement on any value. One distinguished correct node, the King, can decide on the value that all correct nodes have to adapt. The same authors showed that $t+1$ rounds suffice to establish agreement on any input value in the presence of Byzantine nodes. A similar idea was used in [114], where a distinguished node, the Jack, proposed the value that all nodes should adapt.

Algorithm 3.6 presents our method in pseudocode. It is divided into three routines: In the first routine, every node has an input that is broadcast to every other node. Each node sorts the received messages in increasing order and stores them in a local array R. It picks a local approximation of the k^{th} smallest value from R and sets it to be the new input value. In the second routine, every node broadcasts its selected k^{th} smallest value and stores the received values in a sorted array. Then, all nodes exchange their interval bounds to determine the interval in which the k^{th} smallest value should lie from their perspective. The nodes also pick the median of the corresponding array to be their local guess for $S[k]$. The third routine is where the consensus is established. We use the Phase King Algorithm to make the nodes agree on one of the local guesses from Routine 2. Hereby we assume that there are $(t+1)$ predetermined King nodes known to each of

the correct nodes in the system, and each such King is assigned to exactly one phase of Routine 3 in the algorithm.

Throughout the algorithm, we assume that the correct nodes know the total number of nodes n and the upper bound on the number of Byzantine nodes t present in the system. Since Byzantine behavior is arbitrary, we also have to consider the case where some Byzantine nodes decide not to send any value to a correct node in the first routine of the algorithm. Such a correct node would have to choose the approximation of the k^{th} smallest value from a smaller set with fewer Byzantine nodes, thus not satisfying the required approximation for the k^{th} smallest value. One possibility to prevent this case is to fill up the array R with dummy values which are assumed to be worst-case input values, i.e., all smaller than $S[1]$. We can however reach a better local approximation of the k^{th} smallest value by adjusting the number of Byzantine nodes and choosing the k^{th} smallest value directly from the smaller interval. We therefore define $f \leq t$, which denotes the number of values that are suspected to be Byzantine in the array of received values R. We assume that each correct node receives $n - t + f$ values in a round and can calculate f since it knows n and t. Note that this number f depends on the node and the communication round since Byzantine nodes can deviate arbitrarily from the protocol.

Algorithm 3.6 The k^{th} Smallest Value Algorithm

Routine 1: Choosing values close to the k^{th} smallest value

Input: input value x of node v
Output: new input value x^* in the vicinity of the k^{th} smallest value
every node v executes the following commands :

1: Broadcast x
2: Receive input values from every other node, store all values in the sorted array R
3: $x^* :=$ median of the subarray $R[k, k + f]$
4: **if** $x^* \leq R[f]$ **then** ▷ k is too small and $R[k]$ can be an arbitrarily small Byzantine value
5: $\quad x^* := R[f + 1]$
6: **else if** $x^* > R[n - t]$ **then** ▷ k is too large and $R[k]$ can be an arbitrarily large Byzantine value
7: $\quad x^* := R[n - t]$
8: **end if**
9: **return** x^*

Algorithm 3.6 The k^{th} Smallest Value Algorithm, continued

Routine 2: Determining interval with the actual k^{th} smallest value

Input: x^* from Routine 1
Output: trusted interval T for every node and a guess for the k^{th} value s_k
every node v executes the following commands :
10: Broadcast x^*
11: Receive new input values from all other nodes and store them in the sorted array R
12: Broadcast$(R[f+1], R[n-t])$
13: Receive bounds $(R[f+1], R[n-t])$ from all other nodes
14: **for** every new input value x^* that is in at least $n-t$ intervals $\big[R[f+1], R[n-t]\big]$ **do**
15: add x^* to the sorted array T and call it the trusted array
16: **end for**
17: Guess for the k^{th} value $s_k :=$ median(T)
18: **return** trusted array T, guess for the k^{th} value s_k

Routine 3: Phase King algorithm for the k^{th} smallest value

Input: guess for the k^{th} value s_k, trusted array T
Output: consensusValue
19: **for** phase $i = 1$ to $t+1$ **do**
Communication Round:
20: Broadcast(guess for the k^{th} value s_k)
21: receive guesses x from all other nodes
22: **if** some value x is received $\geq n-t$ times **then**
23: Broadcast("propose x")
24: **end if**
25: **if** some "propose x" received $> t$ times **then**
26: guess for the k^{th} value $s_k := x$
27: **end if**
King Round (only the King node executes this round):
28: kingValue = guess for the k^{th} value s_k
29: Broadcast("suggest kingValue")
Decision Round:
30: **if** $s_k ==$ kingValue or kingValue $\in \big[T[min], T[max]\big]$ **then**
31: Broadcast("support kingValue")
32: **end if**
33: **if** "propose x" received $< n-t$ times and "support kingValue" received $> t$ times **then**
34: $s_k =$ kingValue
35: **end if**
36: **end for**

3.3.1 Correctness of the Algorithm

In this section, we will prove the correctness of Algorithm 3.6 and show that the algorithm performs optimally in the proposed model.

Theorem 3.7 (Correctness and Validity)**.** *Algorithm 3.6 achieves Byzantine agreement in the presence of $t < n/3$ Byzantine nodes with a valid consensus value according to Definition 3.1.*

Theorem 3.8 (Termination and Optimality)**.** *Algorithm 3.6 terminates in $O(t+1)$ rounds with message complexity $O((t+1)n^2)$ and achieves an optimal approximation to the k^{th} smallest value.*

3.3.2 Proof of Theorem 3.7

We start by considering Routine 1 and 2 of the algorithm. These are preprocessing steps that force all correct nodes to choose their new input values such that they satisfy the desired validity conditions of the algorithm. Recall that the values in all arrays and subarrays are sorted.

Lemma 3.9. *After the first routine of the algorithm, every node has chosen a new input value that is inside the interval*
$$\Big[S\big[k - \lceil f/2 \rceil\big], S\big[k + \lfloor f/2 \rfloor\big]\Big] \subseteq \Big[S\big[k - \lceil t/2 \rceil\big], S\big[k + \lfloor t/2 \rfloor\big]\Big].$$

Proof. Every node selects its input value as the median of its local subarray $R[k, k+f]$. It is sufficient to show that $S[k]$ is inside any such interval. This is because the algorithm chooses the median of this interval as a guess for $S[k]$. Therefore, it ensures any value from the interval to be at most $\lceil f/2 \rceil \leq \lceil t/2 \rceil$ positions away from $S[k]$.

To prove that $S[k]$ is inside the interval we assume that the Byzantine nodes choose values smaller than $S[k]$. Values larger than $S[k]$ do not influence the position of the k^{th} value. Every value smaller than $S[k]$ shifts the k^{th} value by one position and f Byzantine nodes can shift the k^{th} value by at most f positions. This way, $S[k]$ will always be inside the chosen interval $\big[S[k], S[k+f]\big] \subseteq \big[S[k], S[k+t]\big]$. By the observation above the chosen median of the interval will be at most $\lceil f/2 \rceil$ positions away from $S[k]$. □

This lemma shows that the new input values x^* which are generated by Routine 1 are valid values according to Definition 3.1 for $k \in \big[\lceil t/2 \rceil + 1, n - \lfloor 3t/2 \rfloor\big]$. Other values of k will be considered separately in Section 3.3.3.

Lemma 3.10. *In Routine 2, the nodes decide on a trusted interval T which is a subinterval of*
$$\Big[S\big[k - \lceil t/2 \rceil\big], S\big[k + \lfloor t/2 \rfloor\big]\Big].$$

Proof. Every correct node cuts off the t rightmost and leftmost values of R. The received values from correct nodes are valid values according to Lemma 3.9. There are at most t Byzantine nodes, and therefore also at most t values that are either too large or too small. Step 12 of Routine 2 would remove such values. A value is added to T if it is contained in at least $n-t$ intervals, at least $n-2t$ of which came from correct nodes. Thus, all values in T are valid according to Definition 3.1. □

Lemma 3.11. *The interval T is non-empty for each correct node.*

Proof. We know that all correct nodes will receive the same $n-t$ correct values. Some of the nodes might remove at most t largest and smallest entries after sorting the values. Therefore, at least $n-3t > 0$ central values are left inside the interval T for each correct node. Moreover, this implies that the median of all correct values x^* from Routine 2 will be inside every correct interval T. □

The analysis of Routine 3 is similar to the analysis of the Jack algorithm proposed in [114]. We emphasize the important points of the analysis in the following part.

Lemma 3.12. *If the King adapts the proposed value from Step 23, it will be accepted by all nodes.*

Proof. Only one value x can be proposed simultaneously in Routine 3 by correct nodes. Assume there is another value y that is proposed by some correct node. From any $n-t$ messages that a node received in Step 20, at least $n-t-f \geq n-2t$ values were broadcast by correct nodes. If two correct nodes propose two different values x and y in Step 23, there must have been $2(n-2t) = 2n-4t > n-t$ correct nodes which broadcast either of the values in Step 20. This is a contradiction since each correct node broadcasts only one value. If the King adapts the proposed solution, it has received the proposals from more than t nodes, i.e., at least one correct node. Each such correct node saw other guesses for the k^{th} smallest value x at least $n-t$ times. This means that at least $n-2t > t$ correct nodes broadcast the value and will support the kingValue in Step 31 of Routine 3. □

Note that the trimming procedure in Routine 2 may also cut off correct input values. Nevertheless, the chosen median guess is not cut off in sufficiently many correct intervals. We will show that the median of the trusted array T is inside the trusted interval of at least $(t+1)$ correct nodes.

Lemma 3.13. *If the correct King proposes its own value, all correct nodes will agree on this value.*

Proof. In Routine 2, any correct node decides on a value that is the median of the array of values that it has seen in at least $n - t$ bounds. f of the received bounds could have been malicious, while $n-t-f > t$ of the bounds came from correct nodes. This way, also the value of the King was inside at least $t + 1$ correct bounds. The corresponding nodes will support the kingValue in step 31 of Routine 3. □

In the next part, we will show that the decision value is valid according to Definition 3.1 and also satisfies the standard validity conditions from Section 2.

Lemma 3.14 (Interval Validity)**.** *The decision value of all nodes is a valid value in the sense of Definition 3.1.*

Proof. By Lemma 3.13, the correct kingValue will be accepted by all nodes. The nodes might however have established agreement in one of the previous rounds. We need to show that any value that was accepted by all nodes is a value that was inside at least one trusted interval of a correct node. Any value that was adapted in Step 34 of Routine 3 has been supported by more than t nodes in Step 31, i.e., by at least one correct node and thus was inside its trusted interval. Since any value in the trusted interval of correct nodes is correct, the accepted value must have been correct as well. □

Lemma 3.15 (Any-Input Validity)**.** *The algorithm satisfies Any-Input validity.*

Proof. To show that Any-Input validity holds, we need to consider the first routine of the algorithm. There, the correct nodes choose their new input value. This new input value is a median of values that a node received from all other nodes, i.e., a valid value according to the definition of Any-Input validity. In the next two rounds, the nodes establish agreement on the new input values, where they choose a value that is inside an array of some of the nodes. This value must therefore have been suggested by at least one, possibly Byzantine, node, which proves the statement. □

Lemma 3.16 (All-Same Validity)**.** *Algorithm 3.6 satisfies All-Same validity.*

Proof. For All-Same validity assume that all correct nodes have the same input value x. In the first routine, the Byzantine values will be either equal to x, or they will be not considered in Step 3. Therefore, the new input values x^* in Routine 2 must be equal to the value x, i.e., $x^* = x$. If a Byzantine node chooses a value unequal to x^* in Routine 2, it will land on the right or the left side of the sorted array, and will thus be outside any

interval bounds $(R[f+1], R[n-t])$ making the nodes set $s_k = x^* = x$. All correct nodes will propose x in the third routine, which will immediately lead to agreement. □

3.3.3 Proof of Theorem 3.8

In this section, we prove that the algorithm terminates after a finite number of rounds with an optimal approximation for the k^{th} smallest value.

Lemma 3.17 (Termination)**.** *For $t < n/3$, Algorithm 3.6 requires $O(t+1)$ rounds of communication and terminates with a valid value.*

Proof. The algorithm terminates after sufficiently many nodes have accepted the King's value. Since there are $(t+1)$ predetermined Kings at the beginning of the algorithm, there will be one King that is not Byzantine in at least one of the phases of the algorithm. By Lemma 3.13, all correct nodes will accept the King's value because it is inside the interval of every correct node. This way, all correct nodes will decide on a valid value after at most $(t+1)$ phases. In the case when a Byzantine node proposes a valid value to sufficiently many correct nodes, the algorithm might establish agreement in one of the earlier phases. □

The next lemma shows that Algorithm 3.6 achieves the best possible approximation for the k^{th} smallest value.

Lemma 3.18 (Optimality)**.** *Algorithm 3.6 finds the best possible approximation for the k^{th} smallest value.*

Proof. For $k \in \left[\lceil t/2 \rceil + 1, n - \lfloor 3t/2 \rfloor\right]$, each correct node chooses in Routine 1 of Algorithm 3.6 the best possible approximation to the k^{th} smallest value according to the proof of Theorem 3.4. In the next steps, the decision value is chosen as a value inside the bounds of all correct node values. Therefore, the decision value is between the smallest and the largest approximation of the k^{th} value, and gives a value that is at most $\lceil t/2 \rceil$ positions away from $S[k]$.

In the case where k is outside of the interval $\left[\lceil t/2 \rceil + 1, n - \lfloor 3t/2 \rfloor\right]$, Algorithm 3.6 performs differently. For $k \leq \lceil t/2 \rceil$, each node in the algorithm needs to choose its guess s_k as the $(f+1)$-st value of each node. For $k \in [n-t-f+1, n-t]$, each node chooses the $(n-t)$-th value. According to Theorem 3.5, the local values of the nodes are chosen optimally. With these values, the algorithm will achieve the best possible approximation. □

Thus, Algorithm 3.6 also finds the best approximation for the k^{th} smallest values outside the interval $\left[\lceil t/2 \rceil + 1, n - \lfloor 3t/2 \rfloor\right]$.

Lemma 3.19 (Message Complexity)**.** *The message complexity of Algorithm 3.6 is* $O\big((t+1)\cdot n^2\big)$.

Proof. In the first two routines of the algorithm, the nodes exchange a constant amount of values with each other node which gives an upper bound of $O(n^2)$ messages for the first part. In Routine 3, all nodes exchange their messages with all other nodes in each of the $t+1$ phases. This gives a message complexity of $O\big((t+1)\cdot n^2\big)$ for the last rounds and also the total message complexity. □

It should be noted that the problem of finding the k^{th} smallest value can also be solved using Interactive Consistency (IC) [52, 100]. In this problem, all nodes have to agree on the same vector of n values among which $n-t$ have to be equal to the input values of each of the correct nodes. Each node can then choose the k^{th} smallest value in this vector as its decision value and thus find an optimal approximation to the k^{th} smallest value. IC protocols either need to rely on witness techniques, e.g., the Reliable Broadcast [29, 112], or require parallel execution of the Byzantine Agreement Protocols for each of the input values in order to guarantee that Byzantine nodes cannot send around different values to different nodes. The witnessing technique requires exponential message complexity [52]. The parallel execution of an Agreement Protocol increases the message complexity of a given algorithm by a factor of n, for the King algorithm this leads to message complexity in the order of $O(n^4)$. In contrast to this method, our algorithm shows that it is possible to agree on a value with special requirements, such as the k^{th} smallest value, without increasing the time or message complexity of the standard multivalued Byzantine agreement protocols.

This concludes the analysis of Algorithm 3.6 and shows that the algorithm performs best possible in the distributed setting. In the next part, we will apply the idea of Algorithm 3.6 to find the median of all correct nodes in the distributed setting.

3.4 From the k^{th} Smallest Value to the Median

The median can be computed similarly to the k^{th} smallest value. The main difference is that the Byzantine nodes can shift the k^{th} smallest value by broadcasting values that are smaller than $S[k]$. In contrast, the median can be shifted in any direction by broadcasting larger and smaller values than the median itself. As will be shown in the next theorem, each node therefore has to search for the median inside a symmetric interval $R\big[k-\lceil f/2\rceil, k+\lfloor f/2\rfloor\big]$. As a validity condition we require the consensus value to be inside the

interval $\Big[S\big[m-\lceil t/2\rceil\big], S\big[m+\lfloor t/2\rfloor\big]\Big]$, where m is the position of the median of S according to Definition 3.2. This validity condition was first proposed in [114]. Only the first routine of Algorithm 3.6 needs to be adjusted as presented in Algorithm 3.20.

Algorithm 3.20 Distributed Median Algorithm

Routine 1: Choosing values close to the median

Input: input value x of node v
Output: new input value x^* in the vicinity of the k^{th} smallest value
every node v executes the following commands :
1: Broadcast x
2: Receive input values from every other node, store all values in the sorted array R
3: $x^* :=$ median of R
4: **if** $x^* \leq R[f]$ **then**
5: $\quad x^* := R[f+1]$
6: **else if** $x^* > R[n-f]$ **then**
7: $\quad x^* := R[n-f]$
8: **end if**
9: **return** x^*

Theorem 3.21. *In Algorithm 3.20 the nodes agree on a value that lies within the interval* $\Big[S\big[m-\lceil t/2\rceil\big], S\big[m+\lfloor t/2\rfloor\big]\Big]$, *where m is the median of S.*

Proof. As before, we only need to show that every correct node will decide on a median that is within the interval $\Big[S\big[m-\lceil f/2\rceil\big], S\big[m+\lfloor f/2\rfloor\big]\Big]$ of all correct nodes after the first phase of the algorithm. The Byzantine nodes can shift the median in both directions. Note that placing one value at a position before the median and another one after does not shift the median in the array. The worst case is when all f Byzantine values lie on one side of the actual median. In this case, the median is shifted $\lceil f/2\rceil \leq \lceil t/2\rceil$ positions away from its actual position. It is guaranteed that the guess of the k^{th} smallest value s_k is not further away than $\lceil f/2\rceil$ positions from the actual median since every node picks the median of R as its new input value. Thus, all correct nodes will choose their new input value inside the interval $\Big[S\big[m-\lceil t/2\rceil\big], S\big[m+\lfloor t/2\rfloor\big]\Big]$. □

With Routine 2 and 3 as in Algorithm 3.6, the presented Distributed Median Algorithm computes an approximation for the median which is optimal.

3.5 Vector Consensus

3.5.1 Motivation

Vector consensus is a generalization of the one-dimensional consensus where the nodes have more than one input value and the values of single components are comparable. The idea is to determine a representative vector that is close to the vectors of all correct nodes. There are a number of applications that require vector consensus. One example is the distributed facility location problem, where nodes need to minimize their distance to a median location. Another example are voting protocols, where the voters need to determine a representative median voter. Using the definition of the geometric median in multiple dimensions does however seem difficult since there is no explicit formula to compute a median and only iterative solutions provide an approximation.

In this section, we use the Distributed Median Algorithm to generalize consensus to multiple dimensions. Similar approaches can be found in [87, 88, 120]. In these papers, the authors generalize the idea of removing the t leftmost and rightmost values of the sorted array to several dimensions. They therefore compute the convex hull of every $n - t$ nodes, show that the intersection of all such convex hulls is non-empty and determine a central point inside the intersection. The number of possible convex hulls does however become exponential for large t and computing a central point inside the intersection is therefore costly [120]. The papers instead propose approximate algorithms that can also be applied to asynchronous consensus. In this section, we want to derive an exact version of the vector agreement, while restricting our computation to a linear number of rounds in t. We relax the condition of the consensus value from being inside the convex hull to a value that is inside the range of all correct values in each component. In addition, we have to drop the Any-Value validity condition from Chapter 2, while the All-Same validity condition still holds.

3.5.2 Generalization to d Dimensions

First, we introduce a new validity condition for the general case:

Definition 3.22 (Box Validity)**.** *Let $v^1, \ldots, v^{n-t} \in \mathbb{R}^d$ be the input values of all correct nodes. A vector $w \in \mathbb{R}^d$ satisfies box validity, if for each*

component $i \in [d]$ *holds*

$$\min(v_i^1, \ldots, v_i^{n-t}) \leq w_i \leq \max(v_i^1, \ldots, v_i^{n-t})$$

We generalize the one-dimensional case by applying Algorithm 3.20 to compute the median of each coordinate separately.

Algorithm 3.23 Vector Consensus

Input: n input vectors $v^1, \ldots, v^n \in \mathbb{R}^d$
Output: consensusValue $m \in \mathbb{R}^d$
1: **for** $i = 1$ *to* d **do**
2: use Algorithm 3.20 to compute the median m_i of the values $v_i^1, \ldots, v_i^n$
3: **end for**
4: **return** $(m_1, \ldots, m_d)$

Lemma 3.24. *The algorithm terminates in the presence of* $t < n/3$ *Byzantine nodes with a value that is valid according to Definition 3.22.*

Proof. The algorithm can tolerate up to $(n-1)/3$ Byzantine values since every component is considered separately. In every component, the t leftmost and rightmost values are cut off, and the median is computed according to Algorithm 3.20. Assume some Byzantine value was outside of the interval in some other dimension of the vector. It is not possible for the algorithm to determine whether the node that is an outlier in some component is actually Byzantine. Therefore, we do not need to remove this component of the vector from the set of all nodes. This way we can compute the medians in each component without restricting the number of Byzantine nodes. Observe that the computed median is at most $\lceil t/2 \rceil$ positions away from the correct median in each component and is in the range of all correct values in this component. Thus, the value is guaranteed to satisfy box validity. □

Lemma 3.25. *Algorithm 3.23 satisfies All-Same validity.*

Proof. Assume all correct nodes decided for the same vector. Then, the correct nodes will propose the same value to agree on in each iteration of Algorithm 3.23. By Lemma 3.16 the nodes will agree on the same values in each component, and therefore on the same vector in the end. □

With this algorithm, it is possible to make the nodes agree on a vector that is not too far away from all vectors that were proposed by the correct nodes. The number of Byzantine nodes does not change with the dimension of the input vectors, and the number of rounds is bounded by the number of Byzantine nodes.

3.6 Discussion

In this chapter, we presented a variation of the multivalued agreement problem, where the nodes are required to agree on a particular value from the interval of all correct nodes. We showed that no deterministic algorithm can solve this problem in the presence of Byzantine nodes, but can only approximate the value with an accuracy of $\lceil t/2 \rceil$ positions away from the actual value. We proposed an algorithm for consensus on the k^{th} smallest value in the synchronous message passing model that matches this bound. Using the same algorithm, we were able to improve the result of [114] and find the best possible approximation of the median of all correct nodes. While the algorithm performs optimally in the one-dimensional case, the idea seems not to be applicable to find a representative value for multi-dimensional consensus. In a vector space, the ordering of the input vectors is not well-defined and it becomes computationally expensive to find a representative vector in the presence of Byzantine nodes [87, 88, 120]. In the next chapter, we will define a more accurate median rule for a special case of vectors which correspond to rankings of alternatives. We will extend the results from Algorithm 3.23 and show that it is indeed possible to find non-trivial versions of multidimensional Byzantine agreement that can tolerate up to $n/3$ Byzantine nodes.

4

Byzantine Preferential Voting

In distributed machine learning, different data is often collected and owned by different parties, each of which will locally train its own machine learning model. If a new data item needs to be judged, the parties could collaborate in order to make a collective decision. As an example, a hospital may be authorized to use its own collected patient data to train an image recognition model, but not to share that data with other hospitals because of patient privacy limitations. For some critical cases, the hospitals would still want to collaborate and decide on the correct diagnosis together.

In order to obtain a *robust* collective decision, we need to take the following two aspects into account. On the one hand, it is possible that some of the involved parties experience hardware or software difficulties, or simply play dirty. Our decision will be robust if we can withstand even *Byzantine* parties, who are controlled by a single omnipotent adversary trying to maliciously disturb the process. On the other hand, non-Byzantine parties should use all available information to come up with the best possible decision. In standard multi-valued Byzantine agreement algorithms, each party will provide only one input, however, machine learning algorithms usually provide information about the second-best and third-best guess. For example, when doing image recognition in medicine, the result can be a *ranking* of possible diagnoses: glioblastoma $\succ$ metastasis $\succ \ldots \succ$ inflammatory. Such

rankings convey much more information than just the top-ranked alternative (glioblastoma). While the different honest parties might completely disagree on the top alternative, the second alternative might serve as a tiebreaker, and we can therefore hope to receive more meaningful results from the voting process by considering rankings.

4.1 Background and Motivation

In search of a *fair* rule to elect candidates, philosophers and mathematicians started developing various voting mechanisms and rules already at the beginning of the 18th century. In the middle of the 20th century, Kenneth Arrow [8,9] was one of the first to formalize existing rules and analyze possibility and impossibility results in an axiomatic fashion, thereby introducing the field of Computational Social Choice. In this section, we use this formalism in order to show how well Byzantine agreement connects to voting theory.

We start by considering the special case of n voters voting on only two candidates c_1 and c_2. In this setting, each voter (node) ranks the two candidates such that its preferred candidate (input value) is ranked first. A vote for a candidate c_1 means that the voter strictly prefers c_1 to c_2, here denoted $c_1 \succ c_2$. A central authority then applies a *social choice function (SCF)* to a given preference profile in order to determine the winner (decision value), or set of winners in case of a tie. An SCF f can be qualified based on the following properties:

- f is *anonymous* if interchanging two *voters* (swapping their names) does not change the result
- f is *neutral* if renaming the *candidates* (changing their names) does not change the result
- f is *positively responsive* if in a case where the decision is a tie (c_1 is among the winners) and a voter changes its ranking from $c_2 \succ c_1$ to $c_1 \succ c_2$, candidate c_1 becomes the unique winner

One example of an SCF is the *majority rule*. It chooses the candidate that wins most pairwise comparisons against every other candidate. Note that such a winner always exists in elections with two candidates, but not necessarily in the general case with an arbitrary number of candidates. Social choice theory shows that the majority rule satisfies all desirable properties for the special case of voting on two candidates:

Theorem 4.1 (May's Theorem [80]). *For two candidates and any number of voters, the majority rule is the unique SCF that satisfies anonymity, neutrality, and positive responsiveness.*

Interestingly, most known algorithms for binary Byzantine agreement indirectly exploit the properties of May's theorem. Some of them make use of leaders who suggest their decision value to all nodes, e.g., the King and the Queen algorithms [24, 25]. The leader in these algorithms temporarily plays what is known as a dictator in voting theory. Another type of algorithm, e.g., the shared coin algorithm in [14, p. 314], is biased towards one of the outcomes and thus violates neutrality. In general, we can say that most of the proposed algorithms try to use the majority value as the decision value if a majority exists, or an arbitrary input value otherwise, see for example [23, 29]. Such settings may satisfy anonymity and neutrality, but in cases where the correct nodes are undecided, i.e., there is a tie between the two input values, Byzantine nodes have a large influence on the majority value. Thus, if a correct node decides to swap two candidates in its ranking in order to make one of the candidates win, a Byzantine node can perform an opposite swap in its own ranking and return the profile to the previous state. This shows that positive responsiveness cannot be satisfied for these algorithms in the presence of Byzantine nodes.

May's theorem does not apply to the general case with more than two candidates. In fact, the majority rule gives surprisingly bad results for three or more candidates. To illustrate this, let m denote the number of candidates. Assume that $n/2+1$ voters rank the candidates as $c_1 \succ c_2 \ldots \succ c_m$, and that all other voters rank the candidates as $c_2 \succ c_3 \succ \ldots \succ c_1$. In this case, candidate c_1 wins every pairwise comparison according to the majority rule, even though c_2 seems to be the candidate that is approved by more voters.

Moreover, a lot of information is lost when a single winner is sought. When it comes to preferential voting, social choice theory therefore often wants not only the input to be rankings but also the output. More formally:

Definition 4.2 (Social Welfare Function). *A Social Welfare Function (SWF) is a map from a preference profile to a set of consensus rankings.*

For an SWF g, the following three properties are usually considered:

- g is *dictatorial* if there is one distinguished voter whose input ranking is chosen as the single consensus ranking
- g is *independent of irrelevant alternatives (IIA)* if the consensus ranking of two candidates c_i and c_j only depends on the relative preference of these candidates in each voter's ranking, and not on the ranking of some third candidate c_k

- g is *weakly Paretian* if it satisfies the weak Pareto condition [97]: for two candidates c_i and c_j which are ranked $c_i \succ c_j$ by all voters, consensus ranking has to rank $c_i \succ c_j$ as well

In contrast to IIA and weak Pareto, dictatorship is a highly undesirable property in voting theory. Unfortunately, a corresponding result to May's theorem for SWF's on three or more candidates is the famous impossibility result by Arrow:

Theorem 4.3 (Arrow's Impossibility Theorem [8])**.** *If there are at least three candidates which the members of the society are free to order in any way, then every SWF that is weakly Paretian and IIA must be dictatorial.*

From the viewpoint of Byzantine agreement, an SWF should not be dictatorial since one does not want a dictator to be a Byzantine node. Consequently, any reasonable Byzantine agreement protocol must either violate IIA or weak Pareto. We say that IIA or weak Pareto are satisfied in the Byzantine setting if they are satisfied with respect to the input rankings of the correct nodes only. Under this assumption, the IIA condition implies that the consensus ranking should remain the same if the input of every correct node does not change, no matter what the Byzantine nodes do. However, a Byzantine node can pretend to be a correct node but change its ranking in two executions in which the correct nodes have the same inputs. This change may lead to a different consensus ranking and thus violate IIA. For the weak Pareto condition consider the case with two candidates: if every non-Byzantine voter ranks $c_1 \succ c_2$, the consensus ranking should also rank $c_1 \succ c_2$. This corresponds to a well-known validity condition in Byzantine agreement – the All-Same validity: If all correct nodes have the same input value, all correct nodes have to decide on this value. We use the weak Pareto condition to impose a validity rule on Byzantine Agreement with rankings:

Pareto Validity for any pair of candidates c_i and c_j: if all correct nodes rank $c_i \succ c_j$, then the consensus ranking should rank $c_i \succ c_j$ as well.

Given m candidates, Pareto validity can be viewed as All-Same validity applied on each of the $\binom{m}{2}$ pairs of candidates in a ranking. Note that Byzantine agreement on a ranking is at least as hard as binary Byzantine agreement: Consider a case where the nodes agree on the ranking of the candidates $c_3, \dots c_m$ which they rank last, but not on the two first candidates c_1 and c_2. The Pareto condition is then satisfied for every binary relation which contains at least one of the candidates $c_3, \dots c_m$. Agreement in this case is then reduced to binary Byzantine agreement on the two candidates c_1 and c_2, under the All-Same validity condition.

Unfortunately, there is no straightforward way to apply a binary Byzantine agreement protocol to solve Byzantine agreement on rankings. Other than binary relations on two candidates, preference profiles can form cycles, they can for example contain all three relations $c_i \succ c_j$, $c_j \succ c_k$, and $c_k \succ c_i$ which are each preferred by a majority of nodes. The smallest preference profile which produces such a cycle of binary relations is called a *Condorcet cycle*. It contains three rankings $c_i \succ c_j \succ c_k$, $c_j \succ c_k \succ c_i$ and $c_k \succ c_i \succ c_j$ which induce the three relations from above. Simply agreeing on each pair of candidates can thus lead to a circular decision which does not form a ranking. In order to get rid of cycles, one could think of applying the quicksort algorithm on the candidates sorted with respect to the majority. This procedure will however violate Pareto validity: Consider a candidate c_i that Pareto dominates candidate c_j. Assume that the quicksort algorithm compares both candidates to some third candidate c_k first. Then c_j might win against c_k and c_i might lose, thus swapping c_i and c_j in the consensus ranking. This consideration makes the problem of finding a consensus ranking in the presence of Byzantine nodes rather an instance of multi-valued agreement, as we discuss in Section 4.2, which makes the problem both interesting and challenging.

4.2 A Deterministic Algorithm for Pareto Validity

This section focuses on Byzantine agreement protocols for rankings that satisfy Pareto validity. By using a similar idea to single transferable voting [116] and a multi-valued Byzantine agreement algorithm, a ranking satisfying Pareto validity can be obtained in $(m-1) \cdot (t+1)$ rounds: In the first $t+1$ rounds, we let the voters apply the King algorithm [25] in order to agree on the top candidate. After this, every node removes this candidate from its ranking. In the next step, they will agree on the top candidate from the reduced rankings, and so on. While this procedure is simple, the number of rounds depends not only on the number of nodes but also on the number of candidates.

In the following, we present a deterministic algorithm that solves this problem in only $t+1$ phases using the same number of messages. We do this by modifying the King algorithm to broadcast rankings instead of single candidates. For convenience, we assume that a broadcast operation also includes sending a message to oneself. In the proposed algorithm, we select $t+1$ different nodes and assign each of them to one of the $t+1$ phases of the algorithm. Such a node is called the dictator of the corresponding phase. This dictator then suggests its own, possibly adjusted, ranking to all nodes, which will always be accepted if the dictator is a correct node.

This way, dictators decide on the ranking of all pairs of candidates which do not satisfy the Pareto validity. Algorithm 4.4 presents this procedure in pseudocode.

Algorithm 4.4 Byzantine agreement protocol on rankings (for $t < n/3$)

Every node v executes the following algorithm
1: **for** phase 1 to $t + 1$ **do**
Communication Round:
2: Broadcast own input ranking r_v
3: **for** all pairs of candidates c_i and c_j **do**
4: **if** c_i is ranked above c_j in at least $n - t$ rankings **then**
5: Broadcast "propose $c_i \succ c_j$"
6: **end if**
7: **end for**
8: **if** some "propose $c_k \succ c_l$" received at least $t + 1$ times **then**
9: Adjust own ranking r_v according to Lemma 4.6
10: **end if**
11: **if** some "propose $c_k \succ c_l$" received at least $n - t$ times **then**
12: Fix the pair $c_k \succ c_l$
13: **end if**
Dictator Round:
14: Let node w be the predefined dictator of the current phase
15: The dictator broadcasts its ranking $r_{dictator} \coloneqq r_w$
Decision Round:
16: **if** $r_{dictator}$ agrees with r_v in all fixed pairs $c_i \succ c_j$ from Step 12 **then**
17: $r_v \coloneqq r_{dictator}$
18: **end if**
19: **end for**
20: Return r_v

Since we are dealing with rankings, it is not trivial to see that the nodes always will be able to agree on a proper ranking at the end of the algorithm. The following lemmas state that the nodes can adjust their rankings in Step 9 of Algorithm 4.4 in order to guarantee Pareto validity and that the outcome of the algorithm thus will be a proper ranking. For $t < n/4$ Byzantine nodes one can see that the algorithm is correct since the correct nodes will not be able to propose binary relations which form a Condorcet cycle in this case. In order to show that the algorithm can tolerate $t < n/3$ Byzantine nodes a well, we need to exploit the fact that no Byzantine

node can propose relations that form a Condorcet cycle at any point of the algorithm.

Lemma 4.5. *There is no Condorcet cycle that can be proposed by the correct nodes if* $t < n/3$.

Proof. Assume by means of contradiction that the three relations $c_i \succ c_j$, $c_j \succ c_k$ and $c_k \succ c_i$ were each proposed by at least $t+1$ nodes in Step 4 of Algorithm 4.4. Each binary relation was proposed by at least one correct node who must have seen $n-t$ nodes having a ranking with such a pair.

Let t_1 be the number of all Byzantine nodes who proposed $c_i \succ c_j$, t_2 the number of those who proposed $c_j \succ c_k$ and t_3 those nodes who proposed $c_k \succ c_i$. Further, let $t_{1\cap 2}$ denote the number of Byzantine nodes who proposed $c_i \succ c_j \succ c_k$. The following inequality then holds: $t_1 + t_2 - t_{1\cap 2} \leq t$.

The number of correct nodes who proposed $c_i \succ c_j \succ c_k$ is then $(n-t-t_1)+(n-t-t_2)+t_{1\cap 2}-(n-t) = n-t-t_1-t_2+t_{1\cap 2}$. The number of correct nodes who proposed $c_k \succ c_i$ is $n-t-t_3 \geq n-2t$. However, the two sets must have a nonempty intersection, since

$$n-t-t_1-t_2+t_{1\cap 2}+n-2t-(n-t) = n-2t-t_1-t_2+t_{1\cap 2} \geq n-3t > 1.$$

Therefore, there must be at least one correct node who proposed $c_i \succ c_j \succ c_k$ and $c_k \succ c_i$ simultaneously. This is a contradiction. □

Note that by the properties of the King algorithm, no two opposite binary relations can be proposed in Step 4 simultaneously. Lemma 4.5 additionally shows that a Condorcet cycle cannot be proposed in Step 4 and that all proposed pairs can form a ranking. It remains to be proven that the nodes will always be able to adjust their rankings to incorporate the proposed pairs.

Lemma 4.6. *In Step 9 a correct node will always be able to incorporate the proposed pairs into its own ranking.*

Proof. This is constructed based on the following strategy: Divide the candidates into two sets. The first set contains all candidates which appear in at least one of the pairs proposed by the $t+1$ nodes in Step 9. This set of nodes will be ranked first. The second set will contain all candidates for which the node has not received any propose message. These candidates will be ranked second and will be dominated by all candidates from the first set. Next, we can rank all candidates in the first set according to the proposed relations, possibly leaving some pairs of candidates not ranked. In the last step, all candidates which have not been ranked in each of the sets can be ranked by choosing binary relations from the local ranking of the

node. This strategy outputs a ranking of candidates in which all proposed binary relations are satisfied. □

The next lemma summarizes the correctness results of Algorithm 4.4 and states that the consensus ranking will be valid.

Lemma 4.7. *At the end of Algorithm 4.4 all nodes will have agreed on the same ranking which additionally satisfies Pareto validity.*

4.3 Kemeny Median with Byzantine Nodes

Weakly Paretian voting rules are often not sufficient to pick a fair ranking from a set of individual preference rankings. In search of the best possible consensus ranking, we have to add restrictions on the voting rules without violating the known impossibility results of Arrow [8]. This leads us to majoritarian SWFs, one of which is the Kemeny rule. In the following, we will introduce this rule and use it to derive a better consensus ranking in the presence of Byzantine nodes. Since Byzantine nodes have an influence on the final ranking, the corresponding solutions can be qualified with respect to their approximation ratio which we define in Section 4.3.1. In Section 4.3.2, we will derive lower bounds on the approximation ratio of the Kemeny median in the presence of Byzantine nodes and further provide a matching upper bound in Section 4.3.3.

Definition 4.8 (Kendall's τ distance [64])**.** *The Kendall's τ distance measures the distance between two rankings r and p on candidates $c_1, \ldots, c_m$ by counting all pairs of candidates on which they disagree:*

$$\tau(r, p) \triangleq |\{(c_i, c_j) \mid c_i \succ_r c_j \text{ and } c_j \succ_p c_i\}|.$$

This metric τ on ballots can be extended to a distance function between a ranking r and a profile $\mathcal{P}$:

$$\tau(r, \mathcal{P}) \triangleq \sum_{p \in \mathcal{P}} \tau(r, p).$$

Definition 4.9 (Kemeny median)**.** *For a given profile $\mathcal{P}$, the Kemeny median is the ranking r which minimizes $\tau(r, \mathcal{P})$.*

The Kemeny median satisfies many nice properties and to some extent guarantees that the chosen ranking is "fair". The most prominent quality is probably *monotonicity*: if voters increase a candidate's preference level, the ranking result either does not change or the promoted choice increases in overall popularity. This quality makes the median solution more robust to

Byzantine behavior. The Kemeny rule is also a Condorcet method, it only depends on the number of voters who prefer one alternative over the other and is reinforcing.

Kendall's τ distance, which is used in the Kemeny rule, essentially captures the nature of multidimensionality in our consensus problem. Although it is not straightforward to properly define dimensions for metric spaces, there exist some widely used definitions such as the equilateral dimension. The equilateral dimension is described by the maximum number of points which lie at equal distance from each other. Using the equilateral dimension makes a lot of sense in many cases, it is for example not difficult to see that the equilateral dimension of a d-dimensional Euclidean space is $d+1$. Here we also use the equilateral dimension in order to argue that by using the Kemeny rule we are actually solving a multi-dimensional consensus problem. For any m, we can construct rankings $r_i, i = 1, \ldots, \lfloor m/2 \rfloor$ at equal distance as follows: r_i ranks every candidate j as the j-th element in the ranking and only swaps the candidates $2i-1$ and $2i$. Any pair of rankings in this construction has the same distance 2 to each other and the equilateral dimension of Kendall's τ metric space is therefore at least $\lfloor m/2 \rfloor$.

4.3.1 Byzantine Setting

The Kemeny median cannot be computed exactly in the presence of Byzantine nodes since they might suggest rankings that have a large distance to the Kemeny median of the correct nodes, thus moving the median ranking away from the actual median. A notion for approximate median rankings is therefore introduced as follows:

Definition 4.10 (α-approximation of Kemeny median)**.** *Let κ be a Kemeny median of a preference profile $\mathcal{P}$. An α-approximation of κ is a preference ranking κ_α satisfying*

$$\tau(\kappa_\alpha, \mathcal{P}) \leq \alpha \cdot \tau(\kappa, \mathcal{P})$$

As an example consider binary agreement ($m = 2$): Here τ counts the number of correct nodes who disagree with the consensus value. Any binary Byzantine agreement algorithm that satisfies All-Same validity will also satisfy $\alpha < n - t - 1$.

Unlike binary agreement, it is not straightforward to see what a Byzantine node would choose as its ranking when the Kemeny rule determines the consensus ranking. Since the input vectors of nodes are rankings, each voter has to propose a strict order between candidates and the corresponding preference relation is transitive. A possible strategy for the Byzantine nodes could then be to choose exactly the opposite ranking of the Kemeny

median of all correct nodes. The following lemma shows that this strategy is optimal.

Lemma 4.11. *The opposite ranking of the Kemeny median always gives a worst possible solution which a Byzantine nodes can choose.*

Proof. Note that the sum of the weights of all edges in a tournament graph produced by the rankings of correct nodes only is $(n-t)\cdot\binom{m}{2}$. The tournament graph formed by a ranking r and the tournament graph formed by the opposite ranking $\bar{r}$ are complementary graphs with respect to the tournament graph of all correct rankings. Consider the weights of all possible rankings. Each ranking that minimizes the Kendall's τ distance must have an opposite ranking that maximizes the same distance, i.e., $\tau(r,\mathcal{P})+\tau(\bar{r},\mathcal{P}) = (n-t)\cdot\binom{m}{2}$. This shows that the opposite ranking of the Kemeny median also is the ranking furthest away from it and can always be chosen as a Byzantine value. □

This strategy works for Byzantine nodes, but such a solution is not unique for most preference profiles. To see this, assume that all correct nodes agree on the preference $c_i \succ c_j$ such that this pair will always belong to the Kemeny median of the correct rankings. Then, the Byzantine nodes can pick either $c_i \succ c_j$ or $c_j \succ c_i$ for their ranking since this strategy does not have any influence on the Kemeny median of all rankings. It is therefore difficult for the correct nodes to detect which of the rankings might have been Byzantine.

4.3.2 Lower Bounds on the Approximation Ratio

In this section, we discuss preference profiles that are vulnerable to Byzantine nodes. The first case is based on reducing the rankings to binary agreement and gives the highest approximation ratio for $t < n/3$. Binary agreement does however assume that there are two groups of voters who completely disagree in their preferences. This is somewhat unlikely in practical situations when m is sufficiently large. In the second case, we therefore exclude such binary instances and provide a lower bound based on Condorcet cycles within a preference profile which converges to the same value for large m. The approximation ratio usually depends on the ratio n/t, which will be denoted k for the sake of simplicity.

For our analysis, we represent the preference profile $\mathcal{P}$ as a weighted *tournament graph*, i.e., a graph where the nodes represent the candidates and weighted edges represent how many voters prefer one candidate to the other. The sum of the forward and the backward edges should be equal to the total number of voters in the corresponding preference profile. The

ranking of a node is a directed Hamiltonian path following the order of the ranking, and all other edges are derived from the transitivity. For any two candidates, we denote the edge between these candidates a *majority edge* if its backward edge has a smaller weight. The backward edge we then call a *minority edge*. A Kemeny median of a weighted tournament graph is the ranking that minimizes the sum of the weights of all backward edges of the graph. Note that rankings restrict the power of Byzantine nodes in the sense that Byzantine nodes only can send transitive tournament graphs where every edge has weight 1.

Lower Bound for the Binary Case We first consider all possible preference profiles, in which the worst case is the binary case. This case corresponds to a class of tournament graphs where the Byzantine nodes can redirect all edges by adding t rankings to the preference profiles of the correct nodes. Theorem 4.13 gives a lower bound for the binary case.

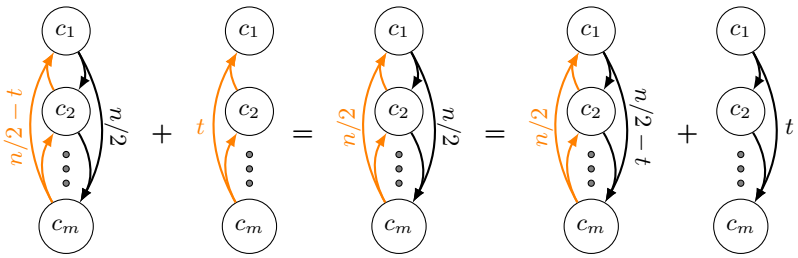


Figure 4.12: Two indistinguishable views on m candidates for binary relations

Note that the labels of the edges correspond to all edges in the same color. The left tournament graph is reached if $n/2$ nodes choose the order $c_1 \succ c_2 \succ \ldots \succ c_m$ and $n/2 - t$ nodes choose $c_m \succ c_{m-1} \succ \ldots \succ c_1$. The right profile can be reached from a profile where $n/2$ nodes choose the order $c_m \succ c_{m-1} \succ \ldots \succ c_1$ and $n/2 - t$ nodes choose $c_1 \succ c_2 \succ \ldots \succ c_m$. The Byzantine nodes can make all correct nodes see the tournament graph in the center by adding t preference vectors $c_m \succ c_{m-1} \succ \ldots \succ c_1$ or $c_1 \succ c_2 \succ \ldots \succ c_m$ respectively.

Theorem 4.13. *There exists a tournament graph corresponding to a preference profile for which the Byzantine nodes may change the edge weights such that no deterministic algorithm can output a ranking which is better than a* $\frac{k}{k-2}$*-approximation of the Kemeny median of all correct nodes, where* $k = n/t$*. For* t *close to* $n/3$*, this gives a* 3*-approximation.*

Proof. This tournament graph is equivalent to binary agreement. Consider therefore one pair of candidates: t Byzantine nodes are only able to change the median, i.e., the majority edge, between these two candidates if they can swap the majority and minority edge by supporting the minority edge with their ranking. Assume the worst case, where the forward and the backward edge both have the same weight $n/2$ after the Byzantine nodes have added their preferences. In this worst case, the tournament graph of correct nodes had the weight $n/2$ for the majority edge. Since the correct nodes will not be able to determine the actual majority edge, they might agree on a minority edge with weight $n/2 - t$ instead. The corresponding approximation ratio is then $\frac{n/2}{n/2-t} = \frac{k}{k-2}$. This result can be easily generalized to m candidates by using opposite rankings.

Figure 4.12 shows a simple generalization of this argument to m candidates and proves that the lower bound of $\frac{k}{k-2}$ holds for all m. □

Lower Bound Excluding the Binary Case In the following, we present another lower bound using Condorcet cycles which can result in ambiguous views as well. We start with one directed cycle formed by three nodes on the tournament graph and assume that every majority edge has a weight of more than $(n + t)/2$, thus discarding the possibility to reduce any pair of forward and backward edges in the tournament graph to binary agreement. The main difficulty in finding a good example comes from the fact that not every tournament graph has an underlying preference profile. We therefore present the necessary properties that preference profiles induce on tournament graphs and show how the best lower bound can be derived.

Start by considering a cycle on the three candidates c_1, c_2, c_3 with majority edges $(c_1, c_2), (c_2, c_3)$ and (c_3, c_1) and assume that this cycle is generated from all rankings, i.e., both the correct and the Byzantine ones. Let the weights on majority edges $(c_1, c_2), (c_2, c_3)$ and (c_3, c_1) be x, y and z respectively. The weights of the forward and the backward edges between any two candidates sum up to the number of all rankings n, such that the weights on corresponding minority edges are $n - x, n - y$ and $n - z$ respectively. Since we exclude cases which reduce to binary agreement and due to the definition of majority edges, we can assume $x \geq y \geq z > (n + t)/2$ or $x \geq z \geq y > (n + t)/2$. Otherwise, adding t to the minority edge would switch the majority and minority edge, e.g., for $x \leq (n + t)/2$ we would get $n - x + t \geq (n - t)/2 + t = (n + t)/2 \geq x$.

In order to calculate the weights of the worst case, we need to consider some additional properties of tournament graphs. One important property is that all weights satisfy the *triangle inequality*, which states that for any

triplet i, j, k it holds that

$$w_{i,j} + w_{j,k} \geq w_{i,k},$$

where $w_{i,j}$ denotes the weight on the directed edge (c_i, c_j). According to the triangle inequality and the lower bound on $x + y + z$, the edges of the cycle c_1, c_2, c_3 satisfy

$$3/2 \cdot (n + t) \leq x + y + z \leq 2n.$$

Note that Byzantine nodes can only change the tournament graph by adding new edges which also satisfy the triangle inequality. The worst case holds if all Byzantine nodes broadcast the same ranking. With six possible rankings on three candidates, we can formulate the worst case as an optimization problem: Let C be the tournament graph formed by all nodes and C_{-b} the tournament graph formed by the correct nodes only. The variable b denotes the ranking added by the adversary, the variable c denotes the ranking chosen by the correct nodes and m is the Kemeny median of C_{-b}. The strategy for the Byzantine nodes then reads as follows:

$$\begin{aligned} \max_{b} \min_{c} \quad & \frac{\tau(c, C_{-b})}{\tau(m, C_{-b})} \\ \text{s.t.} \quad & x + y + z \leq 2n \\ & (n + t)/2 < z, y \leq x \end{aligned} \tag{4.1}$$

The optimal strategy for the correct nodes can be computed by considering all six strategies b of the Byzantine nodes. This gives the following results:

- When $y - z > t$, the optimal strategy is to choose the ranking $c_1 \succ c_2 \succ c_3$, which corresponds to the Kemeny median of C_{-b} for any b.
- When $z - y > t$, the optimal strategy is to choose the ranking $c_3 \succ c_1 \succ c_2$, which corresponds to the Kemeny median of C_{-b} for any b.

The interesting case is when $|y - z| \leq t$. By calculating all possible subcases of this case, it can be verified that choosing c to be the median of C always will give the best approximation for the median of C_{-b}. For all these subcases, the strategies $c_1 \succ c_2 \succ c_3$ and $c_3 \succ c_1 \succ c_2$ are the rankings with the best approximation ratio of the Kemeny median. In the special case $y - z = 0$, the underlying correct tournaments C_{-b} can however become indistinguishable from each other, thus making both solutions, $c_1 \succ c_2 \succ c_3$ and $c_3 \succ c_1 \succ c_2$, equally possible to be chosen by the correct nodes.

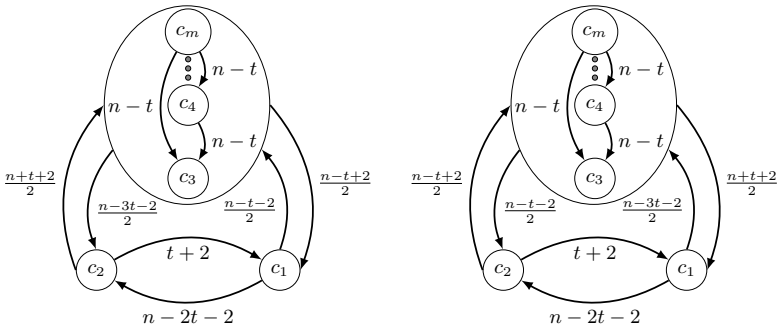


Figure 4.14: Indistinguishable views on m candidates for directed cycles

We have two views which show the profiles of correct nodes only. The left tournament graph results from a profile where $\frac{n-t}{2}-1$ nodes choose $c_1 \succ c_2 \succ c_m \succ \ldots \succ c_3$, $\frac{n-3t}{2}-1$ nodes choose $c_m \succ \ldots \succ c_3 \succ c_1 \succ c_2$ and $t+2$ nodes choose $c_2 \succ c_m \succ \ldots \succ c_3 \succ c_1$. The right tournament graph results from $\frac{n-3t}{2}-1$ nodes choosing $c_1 \succ c_2 \succ c_m \succ \ldots \succ c_3$, $\frac{n-t}{2}-1$ nodes choosing $c_m \succ \ldots \succ c_3 \succ c_1 \succ c_2$ and $t+2$ nodes choosing $c_2 \succ c_m \succ \ldots \succ c_3 \succ c_1$. If the Byzantine nodes add t profiles $c_m \succ \ldots \succ c_3 \succ c_2 \succ c_1$ to the left view, and t profiles $c_2 \succ c_1 \succ c_m \succ \ldots \succ c_3$ to the right view, the resulting profiles become indistinguishable to the correct nodes.

In order to compute the largest approximation ratio, the two optimal strategies can be compared for the six Byzantine strategies. Hereby, the tournament graph C_{-b} can induce new triangle inequalities which restrict the edge weights x, y and z. The largest approximation ratio that we can reach this way holds for the choices $b_1 : c_3 \succ c_2 \succ c_1$ and $b_2 : c_2 \succ c_1 \succ c_3$. The corresponding tournament graphs C_{-b_1} and C_{-b_2} add the inequality $n - 2t > x$ to Equation (4.1). We present the tournament graph for the lower bound in Figure 4.14. The corresponding approximation ratio and the generalization to m candidates are discussed in the next theorem.

Theorem 4.15. *There exists a preference profile containing directed majority cycles in the corresponding tournament graph, for which the Byzantine nodes can add t rankings such that no deterministic algorithm can output a ranking with a better approximation ratio to the actual median than $k/(k-2)$, for m large.*

Proof. Considering a tournament graph formed by one directed cycle of candidates c_1, c_2, c_3, i.e., a directed cycle formed by majority edges. Assume all correct nodes receive a view where $n-2t-2$ nodes prefer c_1 to c_2, where

(c_1, c_2) is a majority edge. Then $(n + t)/2 + 1$ nodes prefer c_2 to c_3 and $(n + t)/2 + 1$ nodes prefer c_3 to c_1. For $n > 3t + 4$, the edge (c_1, c_2) is in the median ranking of all nodes. Since the edges (c_2, c_3) and (c_3, c_1) cannot both be in the median ranking, the nodes have to decide for one of the rankings. In the worst case, one of these two edges was supported by all t Byzantine nodes while the other edge was not supported by any Byzantine node. This leads to two views that are not distinguishable for the correct nodes, as shown in Figure 4.14. The approximation ratio for these views is

$$\frac{n+t+2}{n-t+2} \approx \frac{k+1}{k-1} < \frac{5}{3}$$

An extension to m candidates gives an approximation ratio of

$$\frac{m \cdot n + 2n + t + 2}{m \cdot (n - 2t) + 2n - 3t + 2} \approx \frac{k}{k-2}$$

for large m. □

The received approximation ratio converges to the same approximation ratio as in the binary case for large m, even though we have excluded the binary case from the tournaments. This lower bound underlines the fact that Byzantine agreement on rankings is more complex than binary Byzantine agreement.

4.3.3 Algorithm for Kemeny Median Approximation

In this section, we present a synchronous algorithm for computing a consensus median which matches the lower bound on the approximation ratio presented in the previous section. A simple idea is to use interactive consistency [29, 112]: For $t + 1$ rounds, the nodes exchange all information they have received this far and after the $(t + 1)$-st round they compute the Kemeny median from a set of rankings which they have received often enough. This algorithm guarantees that the set of rankings will be the same for each node and therefore that all nodes will decide on the same ranking. The main drawback of interactive consistency is that it has a large message complexity. The message complexity of this strategy is in $\Theta(mn^t)$ which is exponential for $t \in \Theta(n)$. Also other approaches, such as agreeing on each ranking upfront require the nodes to reliably broadcast their rankings at least once, which results in a message complexity of at least $O(n^3)$ (each node has to forward every received ranking to all other nodes).

Instead of exchanging large amounts of information, we present an approach where we can directly exploit the fact that the Byzantine nodes cannot change a Kemeny median of the preference profile of the correct

nodes by more than a transitive tournament graph with edge weights t. This strategy is presented in Algorithm 4.16.

Algorithm 4.16 Byzantine agreement for the Kemeny median (for $t < n/3$)

Every node v executes the following algorithm
1: broadcast own ranking r_v
2: compute the Kemeny median of the received preference profile, call it m_v
3: apply Algorithm 4.4 with m_v as an input value

Algorithm 4.16 has the same order of round and message complexity as Algorithm 4.4 as stated in the next theorem.

Theorem 4.17. *Algorithm 4.16 terminates within $t + 3$ phases exchanging $O(tn^2 m \log m)$ messages. The computed consensus ranking satisfies the lower bounds from Section 4.3.2 and Pareto validity.*

In the following two lemmas we will prove that the computed solution in Step 3 of Algorithm 4.16 satisfies the bounds from Section 4.3.2.

Lemma 4.18. *In Step 2 of Algorithm 4.16, every correct node chooses a median ranking that matches the bounds from Section 4.3.2.*

Proof. Instead of all nodes in the previous section, we can consider that the Byzantine nodes change just one node's view. Since the number of Byzantine nodes remains the same and the rankings of all correct nodes are received by every node in the synchronous communication model, the Byzantine nodes can in the worst case only reach the lower bound for any correct node, but not exceed it. □

Lemma 4.19. *The computed median ranking by Step 3 of the algorithm satisfies the approximation ratios from Section 4.3.2.*

Proof. Observe that the consensus ranking is derived from a preference profile formed by the medians m_v. Unless some correct node disagrees on an edge in this profile, the edge will be inside the consensus median since Algorithm 4.4 satisfies Pareto validity. This edge may not be inside the median ranking of all correct nodes, but the approximation ratio still satisfies the bounds due to Lemma 4.18.

If the correct nodes disagree on an edge in the preference profile of medians, there was at least one correct node who either chose the opposite edge (binary case) for its median ranking or a different edge in a directed cycle (non-binary case). Consider the binary case first. There, the forward

and the backward edge chosen as the Kemeny median m_v will both satisfy the lower bound, since there is a correct node choosing either of the cases. For the non-binary case, we need to consider directed cycles formed by the median rankings. Every directed cycle in a tournament graph implies that there is a directed sub-cycle formed by three candidates c_i, c_j, c_k. The corresponding preference profile of correct median rankings must contain the three opposite rankings $c_i \succ c_j \succ c_k$, $c_j \succ c_k \succ c_i$ and $c_k \succ c_i \succ c_j$. This, however, implies that all three median rankings could have been derived from the preference profile of the rankings r_v by modifying edge weights by t. The only case from which such a situation can result is when the forward and backward edge weights of each of the three pairs of candidates differed by at most t in the preference profile of rankings r_v. This case, again, is equivalent to the binary case and satisfies the lower bound.

□

Note that the computed median can have a larger Kendall's τ distance to the preference ranking of all correct nodes than any m_v has in the algorithm since the Byzantine nodes can propose their own rankings as dictators in Algorithm 4.4. Such a ranking would still satisfy the lower bound. In the following lemma we will show that the computed median also satisfies Pareto validity.

Lemma 4.20. *Algorithm 4.16 satisfies Pareto validity.*

Proof. Assume, all nodes agree on a preference profile $c_i \succ c_j$. Then, there is a directed edge between c_i and c_j of weight $n-t$. Such an edge always belongs to a Kemeny median, since there cannot be any directed cycle formed by correct nodes only with weights $n - 2t > (n - t)/2$ on all three majority edges due to the triangle inequality. This way, all correct nodes will have the preference $c_i \succ c_j$ in their median ranking m_v. Since Algorithm 4.4 satisfies Pareto validity, the consensus median ranking will also satisfy $c_i \succ c_j$. □

It remains to show that Algorithm 4.16 terminates and has the same order of message complexity as Algorithm 4.4.

Lemma 4.21. *Algorithm 4.16 terminates within $t + 3$ rounds exchanging $O(tn^2 m \log m)$ messages.*

Proof. Note that Algorithm 4.4 terminates after $t + 1$ rounds. Algorithm 4.16 has two additional rounds in which the messages are exchanged. This way, our algorithm terminates within $t+3$ rounds. The message complexity of Algorithm 4.4 is $O(tn^2 m \log m)$, since in every round, each of the n nodes sends a messages of size $m \log m$ to n other nodes. In the additional two

steps of Algorithm 4.16 $2n^2 m \log m$ messages are exchanged which gives the same message complexity $O(tn^2 m \log m)$ for the second algorithm. □

4.4 Discussion

In this chapter, we introduced a new Byzantine agreement problem which extends binary Byzantine agreement to rankings. We showed that rules for choosing a consensus ranking in voting theory fit well with requirements from Byzantine agreement. We further considered a special voting rule, the Kemeny median, for which we provided an optimal Byzantine agreement protocol that can tolerate up to $t < n/3$ Byzantine nodes. It is not clear whether the chosen median rule is the best voting rule for this setting, in particular, since such a rule simply does not exist due to impossibility results in voting theory. It therefore would be interesting to consider a larger pool of voting rules in the future, such as approval voting or the Godgson's rule, and compare them with respect to their Byzantine resilience.

5

The Append Memory Model

Blockchain systems are world-scale systems that are designed to establish a distributed database for transactions via a peer-to-peer communication. Consequently, blockchain research is carried out in the message passing model. In this chapter, we examine blockchain systems from a *shared memory* point of view. In the course of our studies, we found out that shared memory simplifies reasoning for protocols and as such, is a valuable model that can help us understand the fundamentals of blockchains.

In particular, we will introduce a new shared memory model called the *append memory* model. It is a variant of the shared memory model, where data can only be appended (but not modified) in the memory. On the one hand, append memory is stronger than standard shared memory since all written commands appear in the memory. On the other hand, it is also weaker because two concurrent writes to the same register in the shared memory result in only one value being written, whereas in the append memory model such ties cannot be broken. We will show that the append memory model obeys some of the same fundamental impossibility results of asynchronous consensus and that Byzantine agreement cannot be established in less than $t + 1$ rounds in the synchronous model.

For the discussion of blockchain protocols, we will assume probabilistic access to the append memory. This model variant is a clean version of mes-

sage passing proof-of-work blockchains; it allows us to neatly examine one of the main open blockchain research questions: Are DAG-based blockchains superior to tree-based blockchains? In order to understand their difference, we compare Byzantine agreement on a classic tree-based blockchain and a DAG-based blockchain. We show that the DAG achieves an almost optimal resilience (close to $t < n/2$). An orthodox chain, on the other hand, tolerates less than $t < n/(1 + \lambda(n - t))$ Byzantine failures, where λ is the access rate of a node to the memory.

5.1 Model

The considered model consists of n nodes (in the literature often also referred to as processors), $v_1, \ldots, v_n$, that communicate via a shared memory as defined in Section 2.1.3 and aim to establish a relative order on the messages written to the memory. The shared memory consists of n registers, each associated with exactly one node in the system. Each of the registers R_i is unbounded in space and formally supports two operations - R_i.read() and R_i.append(`msg`). We will therefore name this shared memory the *append memory*. The R_i.read() operation can be executed by any node in the system and it returns a complete view of the register R_i. The R_i.append(`msg`) operation can only be executed by the node v_i and it appends the message `msg` to the current state of the memory without removing any previous information from R_i. Since the idea of this memory is to establish a relative order of the messages written to the memory, we assume that a message `msg` from v_i contains some value from this node and a reference to a previous state of the memory that is defined by the underlying protocol. We further assume that the appended messages, just like the registers, are unbounded in space.

In the above definition of the append memory, the n registers can also be viewed as one single register M to which all nodes append their values, with the exception that the single register cannot establish any order on the appended messages. These messages are instead weakly ordered using references to previous states from the underlying protocol. We will therefore also define the M.read() operation, which is going to read the whole memory. It corresponds to executing R_i.read() for all i. Analogously, the M.append($\texttt{msg}_i$) operation appends a new message at any place in M, which corresponds to executing R_i.append(`msg`) in the previous definition. Observe that the single registers R_i may establish a total ordering of all messages corresponding to the node v_i. This ordering can also be incorporated in the single register view by forcing all nodes to refer to their previous appends in the protocol.

In this chapter, we will consider binary Byzantine agreement with synchronous and asynchronous processors according to the definitions in Section 2.1. We will further assume that the considered protocols have to satisfy termination, agreement, and validity (i.e. All-Same validity) or alternatively their weak versions. Note that these definitions do not put any restriction on the access to the memory. In the synchronous and the asynchronous shared memory definitions, the nodes decide locally and independently of other nodes when the memory will be accessed next. In Section 5.5, we will consider an alternative model inspired by the proof of work, where access to the memory is restricted by a Poisson process. We will assume that all nodes can read the memory at any time. An append operation, however, will require a token that is given to the node by some authority who controls the access:

Randomized Memory Access: The access probability to the append memory model for each node v inside the time interval Δ is a Poisson distributed random variable X_v with rate λ. All random variables $X_v, v \in \{v_1, \ldots, v_n\}$ are independent and therefore the access rate to the memory by all nodes is described by the random variable $Y \coloneqq \sum_v X_v \sim \mathsf{Pois}(\lambda n)$

Note that the proposed append memory model deviates from the standard shared memory models in several ways: the append memory cannot order the access threads from different nodes, as the ability to do so would directly imply consensus. This assumption is inherited from the message passing model, where two nodes that received the same two messages might have received them in opposite order. Moreover, the nodes in the append memory have the ability to read the whole memory content with one memory access. This assumption also comes from message passing systems where the nodes receive messages (appends) from other nodes while participating in the protocol and thus reconstruct the whole memory content. Also the randomized memory access is unusual for shared memory models, where the access is usually controlled by the memory itself. Since the append memory withdraws the power of ordering messages from the memory, an access strategy on the protocol side is required in order to be able to establish a weak ordering. of a node to the memory.

5.2 Impossibility of Asynchronous Deterministic Consensus in the Append Memory

The append memory provides a common history of the appended commands to all nodes participating in the consensus. In this section, we will show that

it is impossible to reach consensus in the append memory if the processors in the system are asynchronous and at least one of the processors may crash. With respect to the append memory, the definition of asynchronous nodes says that an arbitrary amount of time can pass between a read and an append operation, meaning that a node might append to an obsolete state of the memory. Dolev et al. also provide a definition for synchronous and asynchronous communication, and show possibility and impossibility results for different communication and processor settings. Their impossibility results hold for the message passing model and rely on the fact that the buffers of the nodes may receive messages in a different order. Such an assumption cannot be made in the append memory since the append memory establishes a common view of the system to all nodes and the states of the system are defined by the point of time at which a node reads the memory.

We will follow the outline of the impossibility proof of Loui and Abu-Amara [78] who showed that it is impossible to reach consensus in the shared memory with read-write protocols. Among other results, the authors provide a proof of impossibility for t-resilient read-write protocols, where the number of read and write operations in the system is unbounded. Our analysis will deviate from the one by Loui and Abu-Amara because our append memory does not allow processors to overwrite the memory cells.

Theorem 5.1. *There exists no t-resilient consensus protocol in the append memory for all $n > 2$.*

5.2.1 Definitions

A *consensus protocol* is a system that consists of n nodes $V = \{v_1, \dots v_n\}$. Each node is equipped with an initial bit which is the nodes' input value. Since the nodes communicate via an append memory, all nodes have read access to the append memory M. We say that the state of a node is defined by the nodes' current value and its last read of the append memory, i.e., if a node last read the memory at time τ, its local view of the memory will be $M(\tau)$. The state of the node from the last read operation is denoted by $s_i = (M(\tau), \mathtt{val}_i)$. A *configuration* C of the system is defined as the set of the states $\{s_1, \dots, s_n\}$ of all n nodes in the system together with the current view of the memory $M(\tau^*)$. We therefore define this configuration to be $C \coloneqq \{s_1, \dots, s_n\} \times M(\tau^*)$. The *initial configuration* of the system consists of the initial bits of the nodes and an empty view of the memory and $M(0) = \{\varnothing\}$, denoted C_0. Each node supports the read and append operations. We will call an execution of a read or an append operation by a node v an *event* e_v. We assume that the nodes are asynchronous. In this system, an event e_v can always be *applied* to a configuration C, if the event

is a read operation. If the event is an append operation, it can only be applied to C if it follows the construction rules of the append memory. Let the current state of node v in the configuration C be $s_i = (M(\tau), \texttt{val}_i)$. An event e_v applied to C at time τ' transitions C to a new configuration $e_v(C)$ in the following way:

(a) e_v is a read operation of the append memory with $M(\tau) \subsetneq M(\tau')$: node v will possibly update its value $\texttt{val}_i$ and transition to a new state $s_i' = (M(\tau'), \texttt{val}_i)$. The corresponding configuration C will transition to the configuration $e_v(C)$ where $e_v(C) = \{s_1, \ldots, s_{i-1}, s_i', s_{i+1}, \ldots, s_n\} \times M(\tau')$.

(b) e_v is a read operation of the append memory at time $\tau' > \tau$, with $M(\tau) = M(\tau')$: node v will not change its state and therefore $e_v(C) = C$ holds.

(c) e_v is an append operation executed at time τ' for the view $M(\tau)$ of the append memory: according to the definition of the append memory, a value from the node v can be appended to an obsolete state of the memory as long as it does not contradict the order of messages of v in the current append memory state $M(\tau^*)$. A state is considered obsolete with respect to a weak ordering defined by the underlying protocol. Let $M(\tau')$ be the state of the append memory after event e_v has been applied to it. Then, the new state of the system is described by $e_v(C) = \{s_1, \ldots, s_n\} \times M(\tau')$.

We say that a configuration C' is *accessible* from another configuration C if there is a sequence $e_1, \ldots, e_j$ of applicable events, such that $C' = e_i\left(e_{i-1}\left(\ldots e_1(C)\right)\right)$. In the following, we define a *computation graph* G for an algorithm $\mathcal{A}$ which describes the accessible configurations: The vertices of G contain all possible initial configurations of the protocol as well as all configurations accessible from these states. There is a directed edge from a configuration C to another configuration C', iff there exists an event e which is applicable to C such that $C' = e(C)$ holds. In this case e will be the label of the edge from C to C'. Note that Property (b) allows self-loops at each configuration of the computation graph. Note that a configuration C' is accessible from a configuration C if there is a directed path from C to C' in G.

Next, we define a *computation* of $\mathcal{A}$ as a (not necessarily finite) sequence of configurations $C_0, C_1, C_2, \ldots$, such that for each pair of consecutive configurations C_i, C_{i+1} there exists a directed edge from C_i to C_{i+1} in the computation graph G. A configuration is said to have a *decision state* if one of the nodes in the configuration has decided on one of the values in $\{0, 1\}$.

A computation *terminates* in a configuration C_j if every correct node in C_j has reached a decision state.

Since algorithm $\mathcal{A}$ solves consensus, in some of the configurations there will be nodes which have reached a decision state. Also, the algorithm has to satisfy the consensus properties from Section 2.1:

- $\mathcal{A}$ satisfies agreement, if every configuration has at most one decision state.
- $\mathcal{A}$ satisfies termination, if for every initial configuration C_0 every computation terminates.
- $\mathcal{A}$ satisfies validity, if for the input configuration where every node has input value 0, i.e., $C_0^{(0)} \coloneqq \{(\varnothing, 0), (\varnothing, 0), \ldots, (\varnothing, 0)\} \times \{\varnothing\}$, every computation terminates with every correct node deciding 0. Analogously, every computation starting in $C_0^{(1)} \coloneqq \{(\varnothing, 1), (\varnothing, 1), \ldots, (\varnothing, 1)\} \times \{\varnothing\}$ terminates with every correct node deciding 1.

Let $\mathcal{C}(C)$ be the set of decision values which are accessible from configuration C under $\mathcal{A}$. We say that C is *bivalent*, if $\mathcal{C}(C) = \{0, 1\}$. C is called univalent if $|\mathcal{C}(C)| = 1$. In particular, C is 0-valent if $\mathcal{C}(C) = 0$ and it is 1-valent if $\mathcal{C}(C) = 1$.

So far, we have only addressed correct nodes in the definition. We call a node correct if for an infinite computation the node takes infinitely many steps. Otherwise, the node is faulty. A computation that that has no events by some node v is called *v-free*. An algorithm $\mathcal{A}$ is 1-resilient, if for any $v \in V$ and any reachable configuration C in G all v-free computations terminate.

5.2.2 Proof of Theorem 5.1

The proof of Theorem 5.1 is organized as follows: We will first show in Lemma 5.2 that for any algorithm $\mathcal{A}$ which satisfies the properties of consensus there is a bivalent initial configuration C_0. In Lemma 5.2 we will show that for any bivalent configuration C and any node v there exists another reachable bivalent configuration C', such that on the directed path from C to C' v performs an event. Finally, we will show how these lemmas can be used in order to establish an infinite computation starting in C_0 in which every correct node performs infinitely many events.

Lemma 5.2. *There exists a bivalent initial configuration of the system if $t \geq 1$.*

Proof. Assume by contradiction that any configuration of the input values is a univalent configuration. Since the consensus algorithm has to satisfy

the validity condition (i.e. if all correct nodes have the same input value, they decide on this input value) there exist 0- and 1-valent initial configurations. In particular, the configuration where all nodes have input value 0 is 0-valent and the configuration where all input values are 1 is 1-valent. Assume that there is one node that can crash in the system. Then, the configuration where all nodes have input values 0 is not indistinguishable from the configuration where all input values are 0 besides the input value of the crashed node, which is 1. Since the nodes have to achieve consensus independent of the crashed nodes and no input configuration is bivalent, both these initial configurations must be 0-valent. We can continue constructing more 0-valent configurations by replacing one input value 0 by input value 1 at a time. Finally, we will have constructed a 0-valent initial configuration that only has input values 1. This is a contradiction to the validity condition. □

Lemma 5.3. *Let C be a bivalent configuration and e_p be an applicable event to C. Let $\mathcal{D}$ be the set of configurations reachable from C that do not contain any events from p. Then, there is a bivalent configuration $C' \in e_p(\mathcal{D})$ reachable from C.*

Proof. Observe first that the event e_p is applicable to every configuration in $\mathcal{D}$: if e_p is a read command, it is trivially applicable to every configuration in $\mathcal{D}$. On the other hand, if e_p is an append command and it can either be appended to the configuration C, or it can be appended to any future configuration as the nodes are asynchronous and there can be an arbitrary delay between a read and an append command in the system.

Let $e(\mathcal{D})$ be the set of all configurations that result from $\mathcal{D}$ by applying the event e_p to them. For the rest of the analysis, assume by contradiction that there are only univalent configurations in $e_p(\mathcal{D})$.

We will first show that $e_p(C \cup \mathcal{D})$ contains both, 0- and 1-valent configurations: By definition, C must contain 0- and 1-valent configurations since it is bivalent. Assume w.l.o.g. that $e_p(C)$ is a 0-valent configuration. Note that all configurations that are reachable from $e_p(C)$ must also be 0-valent. Therefore, there must exist a configuration $D \in \mathcal{D}$ which is 1-valent. Then, the configuration $e_p(D) \in e_p(\mathcal{D})$ is also 1-valent. The corresponding configuration graph is depicted in Figure 5.4.

Further, there exist two configurations D_0 and D_1 in $C \cup \mathcal{D}$ such that $D_1 = e_q(D_0)$ and such that $E_0 \coloneqq e(D_0)$ is 0-valent and $E_1 \coloneqq e(D_1)$ is 1-valent. This follows by an induction argument on the path from C to D. The corresponding situation is depicted in Figure 5.5.

We will differentiate between four cases for the events e_p and e_q:

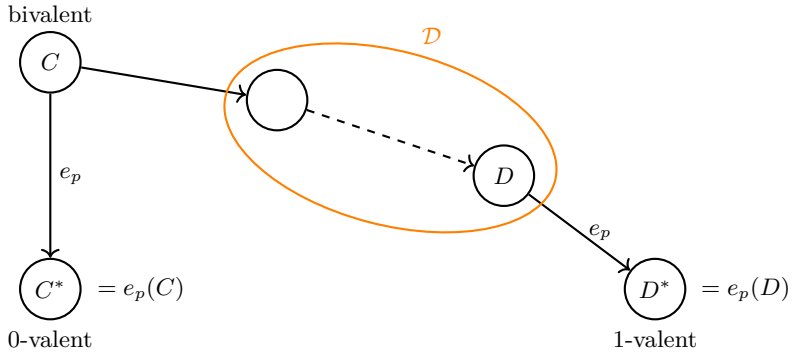


Figure 5.4: Illustration of a bivalent configuration C, where $C^* \coloneqq e_p(C)$ and $D^* \coloneqq e_p(D)$ are respectively 0- and 1-valent configurations reachable from C.

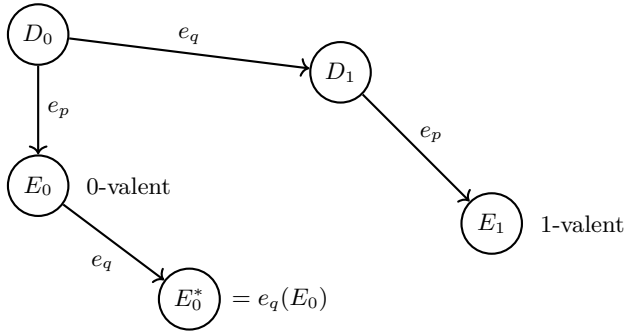


Figure 5.5: Illustration of the proof of Lemma 5.3 with $E_0^* \coloneqq e_q(E_0)$

e_p and e_q are read events: If both events are read events, then the corresponding events are commutative since they do not change the view of the memory. Consider the configuration $e_q(E_0)$. Since E_0 is a 0-valent configuration, $e_q(E_0)$ must also be 0-valent. However, since e_p and e_q are commutative, $e_q(E_0)$ is equal to the configuration E_1, which is a 1-valent configuration. This is a contradiction.

e_p and e_q are append events: We need to show that the append events are also commutative. Note that the append events are consecutive and that neither of the nodes p or q has a read event in between. if both nodes previously read the same view of the memory, they will append to the same view in the memory. Any later read operation will not be able to differentiate which of the appends was written to the memory first. The same property holds if the nodes append to different views of the memory. Since the nodes are asynchronous, they can always append to previous views of the memory and therefore it is not possible to determine the order of the appends. Since the events are commutative, the same analysis applies as in the first case.

e_p is a read and e_q an append event: Note that if e_p is a read configuration, it does not change the view of the memory, i.e., in configurations D_0 and E_0 the view of the memory is identical. Therefore, if e_q is applied to either of the configurations, the configurations of all nodes excluding p are the same in D_1 and $e_q(E_0)$. As the node p might crash during the execution of the algorithm and since the algorithm is deterministic, by applying the same set of events, the remaining nodes must end up in the same configurations independent of whether they started in D_1 or in $e_q(E_0)$. This is a contradiction since the corresponding configurations have different valencies.

e_p is an append and e_q is a read event: In this case we can assume that node q crashes after reaching the configurations E_0 or E_1. The rest of the analysis is analogous to the previous case.

□

Proof of Theorem 5.1 Let $\mathcal{A}$ be a deterministic algorithm that solves consensus in the append memory and can tolerate crash failure. We will prove Theorem 5.1 by showing that there exists a scheduling of events under which $\mathcal{A}$ will not terminate. Note that there exists an initial configuration C_0 that is bivalent according to Lemma 5.2. From this configuration on, we consider a sequence of events in which each node takes infinitely many turns: let $v_1 \dots v_n$ be some ordering of all nodes in the system. By Lemma

5.3 there exists a path from the bivalent configuration C_0 to a bivalent configuration C_1, in which the node v_1 executes an event e_{v_1}. By applying Lemma 5.3, we find another path from the bivalent configuration C_1 to a bivalent configuration C_2, in which the node v_2 executes an event. Lemma 5.3 can be applied infinitely many times in a round-robin fashion to the nodes in the sequence $v_1 \dots v_n$. Since each new configuration C_i is a bivalent configuration, $\mathcal{A}$ will not terminate for any node in the system. This concludes the impossibility proof.

5.3 Consensus in the Append Memory with Synchronous Nodes

In this section, we will discuss possibility and impossibility results for the introduced append memory when the nodes in the system are synchronous. We will first show that the lower bound on the number of rounds needed to solve Byzantine agreement is $t + 1$. We will achieve this bound by slightly adapting the proof of Aguilera and Toueg [3] which was originally introduced for the synchronous consensus in the message passing model with crash failures. The advantage of this proof in comparison to the lower bounds in [49, 52] is that the authors use the notation of univalent and bivalent configuration which we have introduced in the previous section. Other than in the mentioned papers, our lower bound will only hold for the Byzantine model. The previous papers assume that a crashed node can send messages to a subset of the nodes in the system before crashing. This cannot happen in the append memory since the nodes only communicate with one authority (which has control over the memory). Therefore, all values that have reached the memory will be available to all correct nodes after a time interval of Δ. This implies that agreement with crash failures can be solved in the append memory with synchronous nodes within one round only.

In the case of Byzantine failures, the situation becomes different. A Byzantine node can exploit the small asynchrony of Δ in the reads of the nodes such that its append commands will be read by a subset of the nodes in the same round. This gives us the possibility to apply the results from Aguilera and Toueg [3] as we will explain in the next section. In Section 5.3.2, we will provide a matching upper bound which is based on the upper bounds for interactive consistency in the work of Dolev and Strong [49].

5.3.1 Lower Bound on the Number of Rounds

In this section, we will make use of the notation presented in Section 5.2.1. Since we consider a round based algorithm, a configuration C will be asso-

ciated with the configuration at the end of a round. A round is defined as a communication step with the memory, which includes at most one append and one read operation per node. A transition from one configuration to the next can be described by a combination of append and read operations from all n nodes.

We start the proof by stating that there is an initial bivalent configuration C_0. We therefore can use the statement in Lemma 5.2. The difference in the proof is that instead of assuming that a node crashes at the beginning of the algorithm we can assume that this node is Byzantine and does not participate in the protocol.

For the rest of the analysis, we assume that at most one node will exhibit Byzantine behavior in a single round.

Lemma 5.6. *For any round i with $0 < i \leq t$, where t denotes the number of Byzantine nodes in the system, holds: if at the end of round $i-1$ the system was in a bivalent configuration, there is a computation in which at the end of round i the system is again in a bivalent configuration.*

Proof. We will prove this lemma by induction over i. In Lemma 5.2 we showed the base case, i.e., that there exists an initial bivalent configuration.

By the induction hypothesis, we assume that C is the bivalent configuration at the end of round $i-1$. By assumption, at most one node can exhibit Byzantine behavior per round, we call this node b_{i-1}. The power of b_{i-1} in the append memory lies in the fact that it can delay its own messages such that only part of the nodes will see its message in the memory in round i, and the other nodes will only be able to see it with the next read in round $i+1$.

Assume for contradiction that all configurations at the end of round i are univalent. Since C is bivalent, there must exist transitions to 0 and 1-valent configurations. Let the configuration which is reached by a transition where all nodes perform their actions correctly be 1-valent w.l.o.g. We denote this configuration C^1. Moreover, let C^0 be a 0-valent configuration which results through some transition from C. Note that the transitions $C \to C^0$ and $C \to C^1$ only differ in the actions of b_{i-1}, since all other nodes behave correctly and deterministically. Similar to Lemma 5.2, we can construct a sequence of neighboring configurations that differ in the view of one node only such that they have to have the same valency. The construction is the same as in [3]: Let S denote the (possibly empty) set of nodes that see the append of b_i in the memory in the configuration C^0. Consider a configuration C' which results from C^0 by letting an additional node $v \notin S$ see the append of b_i. Note that C^0 and C' are indistinguishable if v fails in round i and therefore both have to be 0-valent. We can continue adding

nodes one by one to S and apply the previous argument repeatedly to show that the configuration C^1 also has to be 0-valent. This is a contradiction to C^1 being 1-valent, and thus there must exist a bivalent configuration in round i. □

Note that Lemma 5.6 implies that Byzantine agreement in the append memory cannot be solved with synchronous nodes in less than $t+1$ rounds. In particular, the lemma shows that there exists a bivalent initial configuration and a t-round computation such that the system ends up in a bivalent configuration at the end of round t. Therefore, the nodes need at least $t+1$ rounds in order to reach a univalent configuration and thus achieve agreement.

5.3.2 A Simple Deterministic Algorithm with Synchronous Nodes

The idea for the synchronous deterministic algorithm in the append memory is similar to the interactive consistency idea in Byzantine Agreement [49]. In interactive consistency algorithms, in every round nodes forward complete views of the system to all other nodes. After $t+1$ rounds, a decision can be made about whether to accept a proposed input value. This results in exponential information exchange. In our case, the views of the nodes are almost the same, since all nodes have access to the append memory. The only differences in the views of the memory can appear through the Byzantine strategy that was described in the previous section. Algorithm 5.7 shows a possible implementation of Byzantine agreement with synchronous nodes.

Algorithm 5.7 Byzantine Agreement with Synchronous Nodes (code for node v)

Input: append memory M, input value $\texttt{val}(v)$

1: **for** round $r = 1, \ldots, t+1$ **do**
2: M.append($\texttt{val}(v), L_{r-1}$), where $L_0 \coloneqq \{\varnothing\}$
3: Wait for Δ time
4: M.read and let L_r be the set of all appended commands in Round r
5: **end for**
6: Let a value $\texttt{val}(w)$ be accepted, if there exists a chain of $t+1$ distinct nodes $v, w_1, w_2, \ldots, w_t$ such that $(\texttt{val}(v), \varnothing)$ is listed in (w_1, L_1), (w_1, L_1) is in (w_2, L_2), …, and (w_{t-1}, L_{t-1}) is in (w_t, L_t)
7: Decide on the majority of all accepted values in $\texttt{val}(w)$

Theorem 5.8. *Algorithm 5.7 solves Byzantine Agreement in the append memory for $t < n/2$ Byzantine nodes within $O(t\Delta)$ time.*

Proof. In order to prove the theorem, we need to show that Algorithm 5.7 satisfies termination, agreement, and validity. Termination is trivially satisfied since all nodes execute the algorithm for $O(t\Delta)$ time and decide. We will show that agreement is satisfied because all nodes will agree on whether to accept each input value or not, i.e., their decision will be based on the same input set. Validity will follow from agreement by showing that additionally to equivalent views, all nodes will accept all correct input values.

Note at first that every correct value will be accepted in Step 6, since there are $n - t > t + 1$ correct nodes: Each correct node will see all correct values appended in the previous round. Therefore, each correct append will be listed in $n - t$ correct appends of the next round, i.e., there will be at least $t!$ chains starting in any correct value which satisfy the condition in Step 6.

Consider next a Byzantine input value b_1. Assume that no correct node contains this value in the set L_1, otherwise, the value will be accepted. b_1 cannot be accepted if no other Byzantine node contains this value in its set L_1. Assume next that the node b_2 contains b_1 in its set L_1. Then, either a correct or some other Byzantine node has to contain this append in their set L_2. Since the chain in Step 6 needs to have $t + 1$ distinct elements and since there are at most t Byzantine nodes, at least one correct node has to contain some append from the Byzantine chain for the value b_1 to get accepted. If one correct node has a value from the Byzantine chain in its set L_i, every correct node will read this set and either accept the value b_1, if $i = t$, or extend the chain by referring to the correct append in the set L_{i+1}. Therefore, a byzantine value will be accepted by the algorithm iff at least one correct node extends the chain of Byzantine appends. □

5.4 Simulation of the Append Memory with Message Passing

In this section, we show how the append memory can be simulated through the message passing model using a simulation similar to the ABD simulation [12]. This completes our theoretical analysis of the append memory and shows that it is a suitable abstraction for different Blockchain and DAG algorithms. In particular, the previous two sections already imply that asynchronous consensus is impossible in the append memory model as well and that the lower bound on the number of rounds for deterministic

Byzantine agreement in the synchronous model is the well-known bound of $t + 1$ rounds. This already shows that our append memory abstraction is not stronger than the results achieved in the message passing model. Here we will show that the message passing model can naturally simulate the append memory model if the nodes sign their messages and assuming that these signatures cannot be forged.

Algorithm 5.9 and 5.10 respectively present the simulation of the append and the read abstractions. Note that the correct nodes need to be *available* at all times, i.e., they always have to respond to append and read messages from other nodes. We will use signatures in order for the nodes to be able to prove that another node sent them a message. We will denote a message $\texttt{val}(v)$ signed by node v by $(\texttt{val}(v))_v$.

Algorithm 5.9 Simulation of M.append() (code for node v)

Input: append value $\texttt{val}(v)$, local memory view M_v

1: Broadcast append$(\texttt{val}(v))_v$
2: **upon** receiving a message from node w **do**
3: Append message from w to M_v
4: Broadcast ack(append$(\texttt{val}(w))_w)_v$
5: **end upon**
6: **upon** receiving ack(append$(\texttt{val}(v))_v)_w$ from $> n/2$ nodes w **do**
7: terminate append operation
8: **end upon**

Algorithm 5.10 Simulation of M.read() (code for node v)

1: Broadcast M.read()
2: **upon** receiving M.read() from node w **do**
3: Send local view M_v to w
4: **end upon**
5: **upon** receiving M_w from $> n/2$ nodes w **do**
6: Append all newly seen values in the local views M_w to M_v and terminate
7: **end upon**

Lemma 5.11. *Algorithm 5.9 correctly simulates an* M**.append(**$\texttt{val}(v)$**)** *operation in the append memory.*

Proof. Note at first that an M.append($\texttt{val}(v)$) operation from a correct node will always reach all other correct nodes. Therefore, all correct nodes

will append a correct value to their local view of M, and node v will receive $> n/2$ acks for its message.

Since Byzantine nodes cannot forge the signatures of the correct nodes, their local view will either contain a message from some correct node or no message at all.

□

Lemma 5.12. *Algorithm 5.10 correctly simulates an* M**.read()** *operation in the append memory.*

Proof. By requesting the views of the memory from more than $n/2$ nodes in the system, a node will receive all append commands added to the local views of the memory by all correct nodes. This is because an append of a node only terminates if $> n/2$ nodes appended the corresponding value to their local view of the memory and by requesting the memory view from $> n/2$ nodes, the append operation will be visible in at least one memory view.

Observe that Byzantine nodes can append multiple values in parallel by sending different messages to different nodes. This is not a contradiction to the correctness of the simulation as such behavior is also possible in the append memory. Since nodes that see two values from a Byzantine party in the append memory cannot distinguish which of the values has been appended first, both values have to be accepted by the correct nodes. The same is achieved in Step 6 of Algorithm 5.10, where all correct nodes accept all values. □

Note that we made use of signatures in order to make sure that a Byzantine party cannot pretend to have received a different value from the correct node than the value sent by the correct node itself. The above algorithms would also work without signatures. In that case, nodes can only append a value to their own local memory, if they have seen it in at least $f + 1$ different views of the memories. Such an adjustment would, however, reduce the resilience of our protocol.

Our analysis shows that the append memory abstracts away the unnecessary communication overhead which often makes the discussion of algorithms in the message passing model difficult and heavy in terms of message complexity. Observe that the size of the local view of the memory increases over with each append operation. Thus, a simulation of an algorithm where all nodes participate, such as Algorithm 5.7, would lead to exponential information exchange.

5.5 Append Memory with Randomized Access

The benefit of a randomized access strategy to the append memory is that algorithms in the permissioned and permissionless settings of Byzantine agreement can be considered. In the first setting, the number of nodes and the corresponding signatures are known to all participants, while in the latter setting, only the upper bound on the fraction of Byzantine nodes is known. In this section, we will focus on the permissioned setting when deriving the bounds. All the presented results can, however, be trivially extended to the permissionless setting as well.

We will first discuss the randomized access to the append memory with respect to the synchronous or asynchronous nodes. We will show that the impossibility result from Section 5.2 can also be applied to this model:

Theorem 5.13. *There exists no deterministic protocol that can solve Byzantine agreement with asynchronous nodes in the append memory with randomized access.*

Proof. Note that the definition of asynchronous nodes states that arbitrary time can pass between any two local operations of a node. The randomized access to the append memory as defined in Section 5.1 only gives out tokens to nodes, such that the nodes can use the token in order to append commands to the memory. As the time between any two operations is unbounded, we can assume that the time between receiving a token and appending a message to the memory is also unbounded. In the worst case, the delays are significantly larger than the append rate to the memory, such that the access order of the memory defined by the random access rule becomes insignificant. Therefore, the proof of Theorem 5.1 can also be applied to this setting. □

The proof of Theorem 5.13 only works because the rate for memory access is independent of the delay resulting from the asynchrony of nodes. This suggests that it is reasonable to consider randomized access to the append memory model together with synchronous nodes. Then, the access rate to the memory can be connected to the maximum delay given between any two operations of the nodes.

In the following sections, we will assume that the input values of the nodes are -1 or $+1$. The decision value will then be determined as the sign of the sum of all accepted values. Note that by the definition of the random access, each node v_i is associated with a random variable $X_i \sim \mathsf{Pois}(\lambda)$ which denotes the expected number of appends by node v_i during Δ. Let $X \coloneqq \sum_{i=1}^{n} X_i$ be the variable denoting how many appends appear

in expectation during the time Δ in the memory. Note that the variable X is also Poisson distributed with $X \sim \mathsf{Pois}(\lambda n)$.

In Section 5.5.1, we will discuss the best possible scenario for the append memory, where each append is equipped with an absolute timestamp. This example will serve as a baseline for the resilience that can be achieved in our model. In Section 5.5.2, we will show that the chain rule, which is often used as a base structure in Blockchain protocols, only tolerates up to $t < n/(1+\lambda(n-t))$ Byzantine nodes. Finally, in Section 5.5.3, we will show that Byzantine agreement in the append memory model with DAGs gives optimal resilience.

5.5.1 Byzantine Agreement with Absolute Timestamps

In this section, we assume that all appends to the memory will be equipped with an absolute timestamp handed out by a central authority upon appending a command to the memory. This way, all appends in the memory will have a unique ordering which is visible to all nodes. Algorithm 5.14 shows how Byzantine agreement can be solved in this model. Although agreement and termination will follow trivially, the validity condition can only be satisfied with high probability. In particular, the probability to satisfy the validity condition will depend on the number of appends to the memory and the difference between the number of correct and Byzantine nodes in the system.

Algorithm 5.14 Byzantine Agreement with Absolute Timestamps (code for node v)

Input: append memory M, input value $\texttt{val}(v)$

```
1: M.read()
2: while there are less than k writes in the memory do
3:     M.read()
4:     upon granted access to the memory do
5:         M.append(val(v))
6:     end upon
7: end while
8: Order all appends by the timestamps
9: Decide on the sign of the sum of the first k appends
```

Theorem 5.15. *Algorithm 5.14 satisfies agreement, termination and weak validity.*

Proof. Algorithm 5.14 satisfies agreement, because the timestamps uniquely determine the first k writes into the memory, and because all nodes have the same view of the memory. By choosing k to be an odd number, the sum of the first k values will be either positive or negative, thus determining the decision value. Termination is satisfied since there eventually will be k writes to the memory such that all nodes will leave the while loop in Step 2.

The validity condition can only be satisfied with high probability, as there is always a negligible probability that the first writes will all be from Byzantine nodes. In order to show validity of Algorithm 5.14, we will consider the sum of all written values as the sum of binomially distributed random variables. For a large number of nodes n, this sum can be approximated by the normal distribution due to the central limit theorem. We can use the tail bounds for the normal distribution in order to finally show that the majority of the k coinflips will be from correct nodes with high probability.

Assume that the validity assumption holds, i.e., that all correct nodes have the same input bit $+1$. W.l.o.g. we can assume that all Byzantine nodes will write the value -1 to the memory. Otherwise, the Byzantine strategy would not be optimal. Note that with probability $p_{corr} = \frac{n-t}{n}$, each append to the memory is from correct node, while with probability $p_{byz} = \frac{t}{n}$, it is Byzantine. Next, we only consider the first k appends to the memory. Then, the probability for each append to be correct or Byzantine will follow the Binomial distribution. Let Y_i be the random variable defining the value of the i-th append in the memory. With above probabilities, we have $\Pr[Y_i = +1] = \frac{n-t}{n}$ and $\Pr[Y_i = +1] = \frac{t}{n}$. We are interested in the probability for the sum of all random variables to be smaller than 0, i.e., the case when a majority of all appended values is Byzantine. Since the nodes in the algorithm wait for at least k coinflips to be appended, the sum of the coinflips converges to the normal distribution $\mathcal{N}\left(k \cdot \frac{n-2t}{n}, k - \left(k \cdot \frac{n-2t}{n}\right)^2\right)$. We can now compute the probability for the Byzantine nodes to reach a negative sum by appending negative values to the memory when given access:

$$\Pr\left[\sum_{i=1}^{k} Y_i < 0\right] < \Pr\left[\sum_{i=1}^{k} Y_i - \mu < \mu\right] < \exp\left(-\frac{\mu^2}{2\sigma^2}\right)$$

where $\exp\left(-\frac{\mu^2}{2\sigma^2}\right) = \exp\left(-\frac{k^2(\frac{n-2t}{n})^2}{2\cdot\left(k-k^2(\frac{n-2t}{n})^2\right)}\right) < \exp(\frac{1}{2} \cdot k \cdot (\frac{n-2t}{t})^2)$. Note that in the worst case, $\#corr - \#byz = n - 2t = \Omega(1)$, $k = \Omega(n \log(n))$

appends to the memory are needed in order to satisfy validity with high probability. If the difference however is equal to $\#corr - \#byz = \Omega(n)$, $k = \Omega(\log(n))$ appended values to the memory are sufficient to satisfy validity with high probability.

□

5.5.2 Byzantine Agreement with Chains

In this section, we will review the results of Garay and Kiayias [55] and Ren [106]. We will show that Byzantine agreement on the chain can also be achieved for $t < n/2$ Byzantine nodes if the nodes use a randomized strategy in order to break ties. Algorithm 5.16 shows an example of such an implementation in the append memory. The idea of the algorithm is to let nodes append their values to the longest chain based on their view of the append memory. Since access to the memory is randomized, with a certain probability, there is a longest chain that consists of a majority of correct input values, such that the decision value satisfies validity. We will differentiate between two tie-breaking rules for the algorithm:

Deterministic tie-breaking: In this rule the correct nodes choose the first longest chain in the memory [55].

Randomized tie-breaking: In this rule, the correct nodes choose one of the longest chains uniformly at random [106].

Algorithm 5.16 Byzantine Agreement with Chains

Input: append memory M, input value $\texttt{val}(v)$

1: M.read()
2: **while** there is no longest chain of length at least k in the memory **do**
3: M.read()
4: **upon** granted access to the memory **do**
5: Let C be the set of the last states in the longest chains of M
6: Choose $c \in C$ according to a tie-breaking rule
7: M.append$(c, \texttt{val}(v))$
8: **end upon**
9: **end while**
10: Decide on the sign of the sum of the first k appends in the longest chain

Both these strategies were mentioned in [55], however, no analysis was given for the second rule. For the deterministic rule, the following upper bound on the number of Byzantine nodes holds:

Theorem 5.17. *Algorithm 5.16 with deterministic tie-breaking cannot solve weak Byzantine agreement for $t \geq n/3$ Byzantine nodes.*

The proof of this theorem is based on the following idea: Since the nodes choose the longest chain according to a deterministic rule, one can assume that all ties will be broken in favor of the adversary. Therefore, one can assume that every append to the memory from a Byzantine node will cause a fork in the chain, i.e., it will append its value to the same append as the last correct node, thus producing two longest chains. With this strategy, every second append to the longest chain will on average be Byzantine. If the Byzantine nodes form a majority, they can change the decision value of the correct nodes even if the validity condition is satisfied for $t \geq n/3$.

Next, we will consider the randomized rule for tie-breaking in Algorithm 5.16. In this case, the previous Byzantine strategy will not be successful, since the correct nodes will only include every second Byzantine append to the memory and the average ratio of Byzantine nodes in the longest chain will be $1/3$.

In the next theorem, we provide a simple bound on the resilience of Byzantine agreement on the chain and show that it is dependent on the rate λ of the Poisson process. The theorem connects the resilience of Byzantine agreement and the access rate of the nodes:

Theorem 5.18. *Algorithm 5.16 with a randomized tie-breaking rule has a resilience of*

$$\frac{t}{n} \leq \frac{1}{1 + \lambda \cdot (n - t)}.$$

That is, for $\lambda \cdot (n - t) = 1$, the resilience is $\leq 1/2$ while for $\lambda \cdot (n - t) = 2$ it is $\leq 1/3$.

Proof. For simplicity, we will restrict ourselves to an average analysis in this proof: The rate $\lambda \cdot (n - t)$ is a measure for how many appends from correct nodes to the memory will take place on average within the interval Δ. In the worst-case scenario, when the delay between any two operations of the correct nodes is δ, appends by correct nodes inside the same interval Δ will be concurrent and therefore generate a fork. Thus, all, but one such correct append can be considered wasted, as it will not be part of the longest chain.

Assume for contradiction that $\frac{t}{n} > \frac{1}{1+\lambda\cdot(n-t)}$. The Byzantine strategy that can be applied in this case is to play the role of a tie-breaker among the concurrent correct appends. This is possible due to the contradiction assumption - the Byzantine party will also have access to the memory in the same interval Δ. The Byzantine party can append its value simultaneously to the first correct append in the longest chain, and thereby prolong the chain by one additional append. Thus, all following correct appends from

the same time interval will append their values to an "outdated" state of the memory, and therefore not make it into the longest chain. With this strategy, the longest chain of size k will have $k/2$ Byzantine values appended to it. Even if all correct nodes have the same input value, $k/2$ Byzantine values inside the longest chain of size k are enough to flip the decision value of the correct nodes. This would violate the validity condition. □

While the above analysis is simple, it is only derived for the average case. Note that, in order to derive a similar bound with high probability, intervals in which the correct nodes have strictly less than $\lambda \cdot (n - t)$ concurrent appends need to be estimated and compared to the number of intervals where the correct nodes have at least $\lambda \cdot (n - t)$ appends and the Byzantine party has also at least one append. Such an analysis has been conducted by Ren [106] who showed that Nakamoto consensus can achieve a resilience of almost $1/2$ if the rate is much smaller than the delay, i.e., when $\lambda \ll \Delta$.

Unlike in the analysis of Ren [106], in Byzantine agreement, we use a fixed interval in order to decide on the agreement value. We would therefore need to take a closer look at what kind of strategies Byzantine nodes can apply just before the decision takes place. We will omit such an analysis for the chain at this point since it is very similar to the analysis of the DAG that will be presented in the next section.

5.5.3 Byzantine Agreement with DAGs

Contrary to the chain, the DAG follows an inclusive strategy: The DAG is a directed acyclic graph that starts at some dummy append, e.g., at the empty state of the memory. All further values that are appended by the nodes only specify the latest seen appends to the memory. That is, if a node sees that another node has appended a value $\texttt{val}(v)$ to the dummy value in the memory, it will list $\texttt{val}(v)$ as its preceding value instead of the dummy append. Listing preceding appends can be viewed as drawing an arrow from the new append to all previous ones which do not have any incoming arrows yet. This strategy generates a directed acyclic graph. Note that the idea of the DAG is very similar to Algorithm 5.7, where all nodes refer to all values they read in the previous round. Since an implementation of rounds requires the nodes to always participate in the broadcast, DAG can be seen as a lighter version of it, where a round consists of parallel appends to the memory. The randomized access to the memory thereby bounds the number of appends from Byzantine parties in each round. Algorithm 5.19 presents a possible implementation of Byzantine agreement on the DAG.

The correctness of Algorithm 5.19 is based on one of the tie-breaking rules in Step 2, such as the heavies chain defined in the Ghost protocol [111]

Algorithm 5.19 Byzantine Agreement with DAG

Input: append memory M, input value $\texttt{val}(v)$

```
1: M.read()
2: while there is no longest (heaviest) containing at least k values do
3:     M.read()
4:     upon granted access to the memory do
5:         Let C be the set of the last states of M, which do not have child
   nodes
6:         M.append(C, val(v))
7:     end upon
8: end while
9: Order the values of the DAG with respect to the longest chain
10: Decide on the sign of the sum of the first k values in the ordering
```

or simply the longest chain [74]. In this section, we are interested in the impact of the worst-case construction of the DAG for Byzantine agreement. In the previous section, it was noted that the worst-case construction of a chain is reached by letting correct nodes generate forks and the Byzantine nodes break the ties. When the nodes are building a DAG, such a construction does not work here, as there will be correct nodes which will include all forked values into the ordering at a later point in time.

The analysis of the DAG therefore focuses on two main issues: The first issue is the rate at which the nodes are appending values. If the rate is too large, the nodes will likely not have the same views when appending to the memory and therefore it will not be possible to determine the global order of the appends. If the rate is small enough, [74, 111] show that w.h.p. there will be a longest chain such that the nodes can decide on an identical view and thus on the ordering of the values in the DAG. The second issue is that Byzantine nodes have the possibility to alter the algorithm by not referencing all values they see in the DAG. While the views of the correct nodes can be identical, a Byzantine strategy can increase the number of Byzantine values among the first k values that are considered for decision.

Lemma 5.20. *In Algorithm 5.19, the views of the DAG upon decision may contain up to* $\Omega(\log(n))$ *additional Byzantine values with high probability.*

Proof. Note that all Byzantine parties can be controlled by one single adversary and that they can withhold their values for a small period of time when the correct nodes are not appending. If the Byzantine nodes apply the strategy for their values among the first k appends, this strategy will not change the ratio between the correct and the Byzantine nodes among

these first writes. Instead, the Byzantine nodes can append a chain of values in the last interval of size Δ just before the decision, thus prolonging the longest(or heaviest) chain and adding their own values to the first k values of the DAG.

We can bound the length of the time interval T during which no correct node appends a value to the memory by the Poisson distribution as follows:

$$\Pr\left[T > \Delta \cdot \log(n)\right] = \exp\left(-\frac{\lambda(n-t)}{t\Delta} \cdot \Delta \cdot \log(n)\right) \leq \frac{1}{n^{\lambda/2}}$$

That is, with high probability there will be an append by a correct node at the end of the interval T.

Next, we calculate the length of the chain that the Byzantine nodes can produce within the time interval T. The size of this chain corresponds to the number of values that Byzantine nodes can insert into the sequence of the first k appends, in addition to the values that are included in the sequence due to the Byzantine rate. Let X denote the Poisson random variable with rate $\mu = \frac{\lambda t}{n}\log(n) \leq \frac{1}{2}\lambda\log(n)$, which corresponds to the Byzantine rate inside the time interval T. We can use the Poisson tail in order to bound the number of Byzantine writes during this time interval:

$$\Pr\left[X \geq \mu + \lambda^2\log(n)\right] \leq \exp\left(\frac{\lambda\log^2(n)}{\mu + \lambda\log(n)}\right) \leq \exp\left(\frac{2}{3}\log^2(n)\right)$$

The above equation states that with high probability, the Byzantine nodes will add less than $2\lambda\log(n)$ values to the memory within a time interval T. □

Theorem 5.21. *Algorithm 5.19 satisfies validity, termination and agreement with high probability.*

Proof. Termination and agreement of Algorithm 5.19 are guaranteed at Step 2 and from the fact that there will be a longest or heavies chain as was shown in [74, 111].

In order to show validity, we consider the same probability distribution as in the proof of Theorem 5.15. Due to Lemma 5.20, the amount of correct writes has to be at least $2\lambda\log(n)$ in order for the correct nodes to satisfy validity:

$$\Pr\left[\sum_{i=1}^{k} Y_i < 2\lambda\log(n)\right] = \Pr\left[\sum_{i=1}^{k} Y_i - \mu < 2\lambda\log(n) - \mu\right]$$

$$\leq \exp\left(-\left(\sqrt{k}\left(\frac{n-2t}{n}\right) - \frac{1}{\sqrt{2k}}\lambda\log(n)\right)^2\right)$$

We next analyse when the above probability becomes exponentially small, which would imply validity with high probability. In the worst case, for $\#corr - \#byz = \Omega(1)$, the number of appends in the memory has to be at least $k = \Omega(\lambda n \log(n))$. In the case $\#corr - \#byz = \Omega(n)$, $k = \Omega(\lambda \log(n))$ values are sufficient. Note that other than in Theorem 5.15, the number of values that are needed for a decision with DAG also depends on the rate λ, which follows from Lemma 5.20.

□

The proof shows that the resilience of Byzantine agreement with DAG is independent of the rate λ. Moreover, this analysis shows that the DAG can tolerate up to $t < n/2$ Byzantine nodes, which corresponds to the optimal bound for Byzantine agreement.

Finally, we would like to emphasize that Nakamoto consensus, unlike Byzantine agreement, does not require finality. In [111], the authors mention that the resilience of Nakamoto consensus on the DAG does not change if the nodes are temporarily asynchronous. The above analysis, in particular Lemma 5.20, shows that this result is not true for Byzantine agreement. In Algorithm 5.19, there is a predetermined number of appends, on which the nodes base their decision. In the case of a temporal asynchrony, the Byzantine nodes could make sure to add more Byzantine values into the set of the first k appends. Therefore, temporarily asynchronous nodes would reduce the resilience of Byzantine agreement on the DAG.

5.6 Discussion

This chapter discusses a new shared memory model, called append memory, that is designed to simplify Blockchain protocols. The strength of the append memory is that it assimilates the local views of the nodes and thus allows simple analysis of the corresponding consensus protocols. In particular, this model abstracts away the sending of messages between nodes in the message passing model which leads to unnecessary complications in Blockchain protocol design. Therefore also discussing the synchronous and asynchronous communication models becomes simpler in this model. On the positive side, this model is not too strong, since it can be simulated in a message passing system as was shown in Section 5.4. The presented impossibility result of asynchronous consensus in Section 5.2.1 suggests that the impossibility of asynchronous consensus is more connected to the shared structure that the nodes construct, rather than the fact that the nodes in

the Nakamoto consensus communicate via a message passing model and thus underlie the FLP impossibility result [53]. In Sections 5.5.2 and 5.5.3 we transferred the known protocols for the Blockchain and BlockDAG into the append memory model. We thereby considered Blockchain protocols for solving Byzantine agreement that have to satisfy agreement, termination, and validity. In this model, we could constructively prove that the resilience of the Blockchain protocol depends on the rate λ, while the resilience of the BlockDAG protocol almost achieves the optimal bound of $t < n/2$ Byzantine nodes. These results suggest that the append memory model is a suitable model to discuss various Blockchain protocols.

Besides solving Byzantine agreement, the append memory model can also be used in order to consider more general problems, such as state machine replication or distributed ledger technologies. Note that for the latter application, often a permissionless setting is considered. In the permissionless setting, the number of nodes in the system is unbounded, instead, the append rate of the values underlies the Poisson distribution. The results for the randomized access in Section 5.5.2 and 5.5.3 were discussed in the permissioned setting only. In fact, these results also hold in the permissionless setting. The upper bounds on the resilience of the protocol in such a setting can therefore be connected to the append rate of the respective nodes (correct or Byzantine). While this model does not affect the definitions of termination, agreement, and validity, it is not clear what the validity condition would mean for infinitely many nodes or whether it would be satisfied at all.

6

Asynchronous Byzantine Agreement and Deep Reinforcement Learning

An important goal of distributed computing is to build computer systems that can tolerate the existence of faulty, or even malicious participants. At the core of such a fault-tolerant system, there is often a problem related to asynchronous Byzantine agreement. Modern Byzantine agreement algorithms will employ variants of asymmetric cryptography, in particular digital signatures. Such signatures can be used to detect malicious behavior, as they may serve as a *proof* that some malicious node sent around contradictory information. If some node has been proven guilty, it can be banished. Once the Byzantine nodes are removed, Byzantine agreement can be solved relatively easily.

Cryptography is however not essential to reach agreement, and there are classic randomized algorithms that do not rely on computational assumptions [23, 31]. Without cryptography, it is difficult to expose the Byzantine participants, since there often will be "word against word" situations. Indeed, without cryptography, there is an *unwritten law* of distributed computing to not even try to uncover the Byzantine nodes, but rather try to break the Byzantine power by randomness and sheer luck. Unfortunately, such non-cryptographic Byzantine agreement algorithms typically require

exponential time to reach agreement. In this chapter, we try to understand to what degree this exponential bound is inevitable.

6.1 Blackboard Broadcast

In this section, we analyze the blackboard matrix (BB-matrix) introduced by King and Saia [69] for broadcasting n^2 coinflips generated by n nodes more reliably. This model is powerful since it keeps the views of every single node as close to the others as possible. As such, this model reduces the number of hidden coinflips from a constant to a polynomial fraction of the total number of coinflips compared to the standard broadcast. We will use the blackboard model in order to generalize the standard reliable broadcast method to a stronger broadcast routine which can assimilate the views of the nodes. We call this routine blackboard broadcast. The blackboard broadcast (BBB) is based on the FIFO reliable broadcast which is defined as follows:

Definition 6.1 (FIFO Reliable Broadcast)**.** *The FIFO (reliable) broadcast defines an order in which the messages are accepted in the system. If a node u broadcasts message m_1 before m_2, then any node v will accept message m_1 before m_2.*

Algorithm 6.2 FIFO Reliable Broadcast

```
1: Broadcast own round r message x(u, r)
2: upon receiving first message x(v, r) from node v for round r do
3:     Broadcast echo(u, x(v, r))
4: end upon
5: if not echoed any x'(v, r) before then
6:     upon receiving echo(w, x(v, r)) from t + 1 nodes w but not x(v, r) do
7:         Broadcast echo(u, x(v, r))
8:     end upon
9: end if
10: upon receiving echo(w, x(v, r)) from n − t nodes w do
11:     if accepted x(v, r − 1) then
12:         Accept(x(v, r))
13:     end if
14: end upon
```

One way in which the FIFO reliable broadcast can be implemented is presented in Algorithm 6.2. Blackboard broadcast strengthens the assumptions of the FIFO reliable broadcast by exploiting it for multiple rounds:

Instead of broadcasting a whole value at once, the local information is divided into small pieces which the nodes broadcast in rounds. After a predefined number of rounds, each node combines all received pieces into one broadcast value from each of the nodes. The following definition describes the conditions that are satisfied by the blackboard broadcast.

Definition 6.3 (Properties of the Blackboard Broadcast)**.** *Given n nodes, among which $t < n/3$ can be Byzantine, BBB is a routine for broadcasting input values while satisfying the following conditions:*

1. *All correct nodes receive the same $n - t$ input values.*
2. *The remaining t values might differ in the last FIFO accepted piece of information.*

The implementation strategy of BBB can be described as follows: a node divides its local value into small pieces, it sends one piece of information in a round. The node does not proceed to the next round until it has collected all pieces of information from $n - t$ nodes in all previous rounds. Algorithm 6.6 describes this procedure.

The correctness of Algorithm 6.6 will be proven next. After this, we will prove that the algorithms satisfies the properties from Definition 6.3.

Lemma 6.4. *At the end of Phase* 1*, matrix* $\texttt{BB}(u)$ *will contain* $n-t$ *columns, each having* n *accepted pieces of information.*

Proof. The matrix $\texttt{BB}(u)$ only contains accepted entries at the end of Phase 1. Each row of the matrix represents a round of communication and each column represents a node. A node only increments its round when it has accepted all pieces of information from the previous rounds from $n - t$ different nodes (including itself). After the last round, the $\texttt{BB}(u)$ matrix holds $n - t$ full columns and t columns where the top part is filled and the rest of the values are unknown. □

Lemma 6.5. *All nodes will finish Phase* 2 *of Algorithm 6.6.*

Proof. Every value of any correct `BB` matrix will eventually be accepted by every correct node. Since the nodes keep updating their `BB` matrix in Step 12, all correct nodes will eventually contain all correct matrices as submatrices and participate in their FIFO broadcast. Once the first correct node accepts $(t + 1)$ matrices, all nodes will eventually accept the matrices, and therefore Phase 2 of the algorithm will eventually be completed. □

Algorithm 6.6 Blackboard Broadcast (BBB)

1: Let $x(v)$ be the input value of node v
2: Split $x(v)$ into r pieces $x(1,v),\ldots,x(r,v)$, such that $x(v) = x(1,v) + \ldots + x(r,v)$

Phase 1: Build BB-matrix row by row

3: **for** round $i = 1$ to r **do**
4: FIFO-broadcast $x(i,v)$
5: Echo $x(j+1,w)$ only if accepted $x(j,w)$
6: Wait until accepted $x(i,w)$ from $n-t$ nodes w
7: **end for**
8: Save all accepted messages as a matrix $\texttt{BB}(v)$

Phase 2: Update BB-matrix

9: FIFO-broadcast $\texttt{BB}(v)$
10: **repeat**
11: Participate in FIFO-broadcast of any $x(k,w)$ as in Step 5
12: Update accepted entries in $\texttt{BB}(v)$
13: Echo $\texttt{BB}(w)$ only if $\texttt{BB}(w) \subseteq \texttt{BB}(v)$ and $\texttt{BB}(w)$ has $n-t$ full columns
14: **until** accepted $(t+1)$ BB-matrices

Phase 3: Decide on values of the nodes

15: Broadcast updated matrix $\texttt{BB}(v)$
16: Wait for $n-t$ other updated matrices
17: Update every entry in $\texttt{BB}(v)$ that was inside of at least one BB-matrix
18: **return** Sum of each column of $\texttt{BB}(v)$

Lemma 6.7. *At the end of Phase* 3*, all correct* BB *matrices share the same* $(n-t) \times (n)$ *submatrix.*

Proof. In order to show that there will be a common submatrix, we need to consider Phase 2 again. There, each node accepts $t+1$ blackboard matrices. Among these matrices, there will be a matrix $\texttt{BB}(v)$ from a correct node. Since the matrices were FIFO broadcast, at least $n-2t$ correct nodes have accepted all values from $\texttt{BB}(v)$ and will therefore broadcast a matrix containing these values in Step 15. Every node also receives matrices from at least $n-t$ correct nodes, $n-2t$ of which will contain all values from matrix $\texttt{BB}(v)$. The statement of the lemma follows from the assumption that $t < n/2$ and Lemma 6.4. □

Lemma 6.8. *At the end of the algorithm, the remaining t columns will only differ in the latest value a node has received.*

Proof. Assume one node has accepted a value in round i. Then at least $n - 2t$ correct nodes have participated in a FIFO broadcast of this value. By the condition in Step 5 these correct nodes must have accepted the value from the same node in the previous round. This means, at least $n - 2t$ correct nodes have accepted the value from the previous round. Therefore, every correct node will accept this value in Phase 3 of the algorithm. □

As the second property in Definition 6.3 states, BBB helps to assimilate the local views of the nodes: assume for comparison that input values ± 1 are broadcast using the standard broadcast abstraction first. If an adversary controls the scheduling, it can make the views of the nodes differ by a total sum of $|t|$ by hiding t values from some correct nodes but revealing this information to the remaining correct nodes. If the correct nodes use BBB to broadcast the same input values, and broadcast pieces of size $1/|r|$ in each round (where r is the total number of rounds), the local views of the nodes will only differ by a sum of at most $|t/r|$, meaning that the adversary can hide proportionally fewer values using scheduling. Note that each node can also broadcast n generated coinflips in n rounds using BBB instead of dividing some random value into small pieces. This strategy is sufficient to generate a shared coin in the crash failure model since the `BB`-matrix will contain $n^2 - n$ coinflips that are seen by all nodes. BBB can however not be used to directly establish Byzantine agreement. This is because Byzantine nodes still control $t(n-t)$ values in the `BB`-matrix. They can therefore make sure that the sum of all values is close to the threshold needed for a decision.

It can also be argued that BBB makes Byzantine nodes detectable: for the correct nodes to be undecided, Byzantine nodes need to cancel the sum of the correct values. Since the values controlled by Byzantine nodes are less than 1/3 of all values in the `BB`-matrix, their sum would need to deviate more from the threshold used for the decision of the correct nodes, thus making them detectable. This detectability was also exploited by King and Saia [67–69], who suggested to repeatedly build `BB`-matrices, until the nodes who expose their Byzantine behavior can be removed from the protocol. They also claim that they only need a polynomial expected number of `BB`-matrices in order to reach agreement. In the following section, we will briefly discuss the two algorithms used in [69] and why detectability of Byzantine nodes as it was performed in this paper is tricky when a Byzantine party controls the scheduling.

6.1.1 Detectability of Byzantine Parties using BBB

In [69] the authors propose two algorithms to solve asynchronous Byzantine agreement in expected polynomial time by detecting Byzantine nodes. They make use of the blackboard broadcast in order to broadcast large randomly generated values, in their matrix each node generates n random values that are either $+1$ or -1. The blackboard broadcast is executed for $m \in \Theta(n)$ rounds. This way, a new matrix $\mathcal{M}$ is constructed, where each column corresponds to a node (like in the BB-matrix), while every row corresponds to one round of blackboard broadcast. The main idea of [69] is to show that in a constant fraction of the rows of $\mathcal{M}$, the sum of the Byzantine values will deviate from the expected value in order to prevent correct nodes from agreement. This deviation is used to add scores to the Byzantine nodes which eventually will lead to a removal of these nodes from the algorithm. In the first proposed algorithm, the authors use spectral methods and assign weights using the top right singular vector of $\mathcal{M}$. In the second algorithm, a brute force search through all submatrices of $\mathcal{M}$ of size $cm \times i$, where $0 < i \leq t$, is performed. If the sum of all values in such a submatrix exceeds a threshold, the weight of the corresponding nodes is increased. In the following, we will discuss a possible Byzantine strategy that has not been addressed in the analysis of [69].

Byzantine Stopping Strategy In the blackboard model, there are t columns that can be hidden by the Byzantine nodes. We can define a Byzantine strategy as follows: the Byzantine nodes can fix some t nodes that will be called *controlled*, wait until most of these nodes reach a certain threshold, and hide all other values from these nodes. They then can adjust their own values such that the sum of all coinflips is close to 0. If the Byzantine nodes see all possible coinflips until the n-th round, it is easy to decide at which point to stop these values. If the Byzantine nodes only see the next coinflip, which can be implemented, we need to argue that w.h.p. the Byzantine nodes will be able to stop a large fraction of these t fixed nodes.

The problem of one node being stopped at a certain point can be described as an 1-dim simple random walk. In order for the controlled nodes to be removed by the algorithm, the Byzantine nodes only need to get the sum of these nodes above the standard deviation of the correct nodes. The deviation of the sum of the correct (not controlled) nodes is $-c'n$ with constant probability. Therefore, each of the t controlled nodes needs to get above a value of $c'n/t$ on average, which is just a constant for $t \in \Theta(n)$.

Let c be a sum that a Byzantine node wants to generate in some controlled node. Let $\tau(c)$ denote the time point, i.e., a row in the BB-matrix, in

which the sum of a node reaches c for the first time. The probability for a node not to reach this sum for the first time after i steps is

$$P[\tau(c) = i] = \frac{c}{i} \cdot \binom{i}{(i+c)/2} \cdot \frac{1}{2^i} \leq \frac{c}{i} \cdot \binom{i}{i/2} \cdot \frac{1}{2^i} \approx \frac{c}{i} \cdot \frac{2^i}{\sqrt{i}} \cdot \frac{1}{2^i} = \frac{c}{i\sqrt{i}}$$

The probability to reach c for the first time after the n-th step is:

$$\begin{aligned} P[\tau(c) \geq n] &= \sum_{i=n}^{\infty} \frac{c}{i} \cdot \binom{i}{(i+c)/2} \cdot \frac{1}{2^i} \leq \sum_{i=n}^{\infty} \frac{c}{i} \cdot \frac{2^i}{\sqrt{i}} \cdot \frac{1}{2^i} = \sum_{i=n}^{\infty} \frac{c}{i\sqrt{i}} \\ &\leq \int_n^{\infty} \frac{c}{x\sqrt{x}} dx = \frac{c}{\sqrt{x}} \Big|_n^{\infty} = \frac{c}{\sqrt{n}} \end{aligned}$$

The largest sum that the Byzantine scheduler can generate for a node with high probability is $c = n^{\frac{1}{2}-\alpha}$, where $0 < \alpha < 1/2$ is a small constant. Plugging this c into the above inequality gives

$$P[\tau(n^{1/2-\alpha}) \geq n] \leq \frac{n^{1/2-\alpha}}{\sqrt{n}} = \frac{1}{n^{\alpha}}$$

and the probability to succeed within the first n iterations becomes

$$P[\tau(n^{1/2-\alpha}) < n] = 1 - \frac{1}{n^{\alpha}}$$

We will next show that almost all correct nodes can be stopped at a value of $n^{1/2-\alpha}$ with high probability. This will help us to find a lower bound on the subset of nodes that can be stopped in a constant fraction of BBB iterations.

Theorem 6.9. *After the first m iterations in the blackboard model, $(1 - 1/n^{\alpha/2})^2 \cdot t$ of the controlled nodes can be stopped at a value of $n^{1/2-\alpha}$ w.h.p., where $0 < \tilde{\alpha} \leq 1/4$ is a small constant.*

Proof. We use the Chernoff bound to show this result. The probability to stop one node within n iterations is $1 - \frac{1}{n^{\alpha}}$. The expected number of stopped nodes at the end of n iterations is then $\mu = t \cdot \left(1 - \frac{1}{n^{\alpha}}\right)$. Let $t \coloneqq \varepsilon \cdot n^{1/2+\gamma}$ for some constants $0 < \gamma \leq 1/2$ and $0 < \varepsilon < 1$. This assumption does not restrict the setting, as Byzantine agreement can be solved in an expected constant number of rounds for $t \in \Omega(\sqrt{n})$. WLOG we can assume that $\alpha \geq \gamma$, otherwise we can replace α by γ in the theorem statement. We next define a random variable X_i to be 1 if a controlled node could be stopped

and 0 otherwise. Let $X = \sum_{i=1}^{t} X_i$ denote the number of stopped nodes. Now we can apply the Chernoff bound with $\delta = 1/(n^{(\alpha+\gamma)/4})$:

$$\begin{aligned} P\left[X < \left(1 - 1/n^{\alpha/2}\right)^2 \cdot t\right] &\leq P[X < (1-\delta)\cdot\mu] \leq \exp\left(-\frac{\delta^2 \cdot \mu}{2}\right) \\ &= \exp\left(-\frac{1}{2} \cdot \left(\frac{1}{n^{(\alpha+\gamma)/4}}\right)^2 \cdot \left(1 - \frac{1}{n^\alpha}\right)\right) \cdot t \\ &\leq \exp\left(-\frac{1}{2} \cdot \frac{1}{n^{(\alpha+\gamma)/2}} \cdot \frac{n^\alpha - 1}{n^\alpha} \cdot \varepsilon n^{1/2+\gamma}\right) \\ &\leq \exp\left(-\frac{1}{3} \cdot \frac{1}{n^{(\alpha+\gamma)/2}} \cdot n^\alpha \cdot \varepsilon n^\gamma\right) \end{aligned}$$

by defining $\tilde{\varepsilon} \coloneqq \frac{1}{3}\varepsilon$, we get

$$P\left[X < \left(1 - 1/n^{\alpha/2}\right)^2 \cdot t\right] \leq \exp(-\tilde{\varepsilon} n^{\alpha/2})$$

□

The above strategy shows that among the controlled t nodes by the adversary, more than a large constant fraction can be stopped in a round. Note that the probability not to stop this fraction of nodes is exponentially small. Therefore, this property also holds for consecutive $m \in O(n)$ rounds with high probability.

We will next show that inside the matrix $\mathcal{M}$, there will be a large submatrix where each entry has a sum larger than $n^{1/2-\alpha}$. We can show this using a similar strategy to Lemma 5.5 in [69].

Theorem 6.10. *Let $\mathcal{M}'$ be the $m \times t$ submatrix of $\mathcal{M}$ controlled by the Byzantine nodes and $c_1, c_2 < 1$ constants that are arbitrarily close to 1. Then, w.h.p., there exists a submatrix in $\mathcal{M}'$ of size $c_1 \cdot m \times c_2 \cdot t$, where each entry has been stopped by the Byzantine strategy.*

Proof. Let x be a variable defining the number of columns in the desired submatrix, and y define the number of rows. Assume that the chosen x and y are maximal, i.e., given x columns, there exists no submatrix of $\mathcal{M}'$ of size $(y+1) \times x$ that satisfies the property of the theorem. Then, due to this maximality condition, the following inequality has to hold:

$$x \cdot m + y \cdot (t - x) > (1 - 1/n^{\alpha/2})^2 \cdot t$$

The idea behind this strategy is the following: if in x columns and $y-1$ rows all entries were stopped by the Byzantine adversary, then there can be no

more stopped entries in this submatrix; Otherwise, x could be extended by one more column. Note that the inequality gives us a possibility to derive lower bounds on the size of the submatrix: every pair of values x and y that do not satisfy the above inequality are a lower bound for the size of the largest submatrix.

Note that the above inequality is violated for any constant fractions of t and m, given that n is sufficiently large. Therefore, the theorem statement holds. □

A large constant fraction of controlled nodes will be stopped at values of at least $n^{1/2-\alpha}$. This way, Byzantine nodes can make sure that a subset of correct nodes has a large sum. Alternatively, the submatrix of the correct nodes will contribute to a large norm of $\mathcal{M}$. While it has been shown in [70] that, with this strategy, the submatrix of the Byzantine nodes will also have a large norm, many questions about which nodes will be detected first by the algorithm remain open.

The analysis of asynchronous Byzantine agreement via the blackboard broadcast is fairly complicated, which makes it difficult to eliminate all possible Byzantine strategies. In the next section, we will focus on possibilities to develop Byzantine agreement algorithms in a more automated way by using deep reinforcement learning and self-play.

6.2 Byzantine Agreement with Reinforcement Learning

In this section, asynchronous Byzantine agreement has been simulated using a neural network as the omnipotent adversary. A Byzantine agent was therefore trained using deep reinforcement learning by letting it control $t < n/10$ nodes. If successful, the trained Byzantine agent should be able to reach the exponential running time of the Ben-Or algorithm (Algorithm 2.3). The purpose of this study is thus to investigate whether deep reinforcement learning can be used to adopt effective tactics to postpone agreement. Before explaining the implementation, we will first introduce the reinforcement learning problem following the descriptions from [115].

Consider an agent interacting with an environment. Assume that the agent first observes the states $s_\tau \in \mathcal{S}$ and then takes an action $a_\tau \in \mathcal{A}$ for which it receives a reward $r_\tau \in \mathbb{R}$. Here, $\tau = 0, 1, 2, 3, \ldots$ denotes the discrete time steps. The agent acts according to her policy $\pi(a|s_\tau)$ which is a conditional probability distribution over the actions given the current state s_τ. The agents' goal is to maximize the *expected (discounted) reward* defined as $R_\tau = \sum_{k=\tau}^{\infty} \gamma^{k-\tau} r_{k+1}$, with $0 \leq \gamma \leq 1$, where γ is the discount factor.

We further assume that the states satisfy the *Markov Property*. That is, the response at time step $\tau + 1$ only depends on the action and state at time τ. The corresponding reinforcement learning task is thus a *(finite) Markov Decision Process (MPD)*. This process is fully described by the one-step dynamics, i.e., the *transition probabilities*

$$\mathcal{P}^a_{ss'} = \mathbb{P}(s_{\tau+1} = s' | s_\tau = s, a_\tau = a)$$

and the expected next reward

$$\mathcal{R}^a_{ss'} = \mathbb{E}(r_{\tau+1} | s_\tau = s, a_\tau = a, s_{\tau+1} = s').$$

We can now define the *action-value function for policy* π as

$$Q^\pi(s, a) = \mathbb{E}_\pi(R_\tau | s_\tau = s, a_\tau = a).$$

In particular, we are interested in the optimal action-value function $Q^*(s, a) = \max_\pi Q^\pi(s, a)$ that satisfies the *Bellman Equation*

$$Q^*(a, s) = \mathbb{E}(r_{\tau+1} + \gamma \max_{a'} Q^*(s, a') | s_\tau = s, a_\tau = a)$$

There are several methods to solve the reinforcement learning task, e.g., via the state-value or the policy iteration. In the following part, we will focus on *Q-learning*.

Q-learning The above Bellman Equation for the Q-function can be solved using an iterative process, which was shown to converge to the optimal value in [40]. Consider an agent in stage k observing s_k, taking action a_k, receiving reward r_k, and observing next state s_{k+1}. The agent can update its action-value function according to

$$Q_k(s, a) = \begin{cases} (1 - \alpha_k) Q_{k-1}(s, a) \\ \quad + \alpha_k [r_k + \gamma \max_{a'} Q_{k-1}(s_{k+1}, a')] & , \text{if } a = a_k \text{ and } s = s_k \\ Q_{k-1}(s, a) & , \text{otherwise} \end{cases}$$

where α_k is a predefined learning rate.

Deep Reinforcement Learning In Deep Reinforcement Learning, a neural network is used to approximate the Q-function [91], i.e., $Q(s, a; \theta) \approx Q^*(s, a)$. To find the parameters one can use the loss function

$$L_k(\theta_k) = \mathbb{E}_{s,a}[(y_k - Q(s, a; \theta_k))^2]$$

with $y_k = \mathbb{E}_{s,a}[r + \gamma \max_{a'} Q(s', a'; \theta_{k-1}) | s, a]$.

For this purpose, we use the DQN algorithm, which is based on Q-learning.

6.2.1 Implementation

To implement our approach, we use the DQN algorithm by Stable Baselines [96] with a 2 hidden layer Multi Layer Perceptron (MLP) network and 64 nodes in each hidden layer. The DQN algorithm uses an ϵ-*greedy* strategy, meaning that a random action is chosen with probability ϵ and, otherwise, the action maximizing the current Q-function is chosen. The DQN algorithm starts with $\epsilon = 1$ and reduces it to $\epsilon = 0.02$ over the *exploration fraction* of the entire training period, then it keeps $\epsilon = 0.02$ constant for the rest of the training. For the environment, we follow the formalism of the Gym-package [95].

In our case, the environment is given by the Ben-Or algorithm. In the following, we will refer to a full run of a Byzantine agreement algorithm as a *game* or an *episode*, and to one time step as *step*. We will refer to the adversary controlling all Byzantine nodes as the *Byzantine agent* or simply the *agent*.

The Ben-Or algorithm does not have distinct rounds and therefore every phase directly corresponds to a step in the game. As in most asynchronous algorithms, the correct nodes only await $n - f$ messages, meaning that they can consist of anything between only those of correct nodes or those of $n-2t$ correct and t Byzantine nodes. The Byzantine agent observes $n-t$ messages from all correct nodes and then decides which values (at most t) she wants to replace with a value of her choice. The correct nodes then receive the modified $n-t$ messages. As we only consider simple broadcast methods, the agent can send a different modification to each correct node. The Ben-Or algorithm relies on the chance that all correct nodes get the same value from the coin, but there are three interesting variations that we will introduce below.

Fair Coin In this case, we study the Ben-Or algorithm stated in Section 2.1.2 where Step 11 is a fair coin that is flipped locally by the nodes. The Byzantine agent observes the (binary) messages sent by the correct nodes. The agent can only send the same modified message-vector to all correct nodes. This implies that her goal must be to let the correct nodes flip a coin, otherwise, all nodes will decide within two steps. After each step, the agent receives a reward if she lets the correct nodes flip a coin. The game is over when a correct node has decided (Step 7 of Algorithm 2.3). $\lfloor n/2 \rfloor + 2t + 1$ correct nodes may flip the same value by chance, in this case, the agent has won the game and receives an additional reward.

Predefined Coin Here, we consider a modification of Step 11 of Algorithm 2.3, where the coin is not random but a predefined value that was

chosen at the beginning of the algorithm where $Pr[x_u = 0] = 1/2$ and $Pr[x_u = 1] = 1/2$. This case corresponds to the bitstring idea discussed in Lemma 2.4. The Byzantine agent observes the value of the coin of the current and the next rounds in addition to the messages sent by the correct nodes. We restrict the view of the bitstring to two rounds since the proof of Lemma 2.4 showed that a look-ahead of two values is sufficient for the algorithm to not terminate. In this case, it is not optimal for the agent to let all correct nodes flip the coin since all of them will receive the same value. The agent is therefore allowed to send different modifications of the messages to different correct nodes. It is important to note that the Ben-Or algorithm does not need to terminate in this setting. However, this does not lead to problems during training as the agent always explores with a certain probability and therefore makes mistakes, which lead to termination of the game. We further differentiate between the following two settings in this case:

Predefined initial value: This is the case, where the initial values of the nodes are predefined. Assume that the first value of the predefined coin is $x_c = 1$. Then, we need to have initial values with a majority of 0's, otherwise, agreement will be achieved in the following Ben-Or round. More precisely, the number of zeros needs to be between $\lfloor n/2 \rfloor + 1$ and $\lfloor n/2 \rfloor + 2t$, such that the agent can let some correct nodes flip a coin, and some enter Step 9 of Algorithm 2.3. Whether the agent sends the majority of the nodes to flip the coin or not depends on the value of the next coin.

Agent selects initial values: This version is largely the same as the above, with the difference that the Byzantine agent selects the initial values of the correct nodes. The agent observes a vector of 0's in the first step and decides for each entry. The agent therefore needs to be informed whether or not she observes the first step.

6.2.2 Training

For training, we mostly use the default values for the parameters of the Stable Baseline's DQN Algorithm [96]. The *exploration fraction* is increased form 0.1 (default) to 0.2, leading to a longer period of exploration. The total length of the training period differs for the various versions. To monitor and evaluate the performance of the learner (agent), we will consider the *learning curve* and the *deterministic validation performance*.

The learning curve shows the (smoothed) rewards the agent achieved in each episode throughout the training. On one hand, the agent should be able to learn, i.e., the rewards should increase with the progression in

the learning process. On the other hand, the agent should show a stable performance, meaning that there should be no large fluctuations in rewards between episodes.

In the following, we will discuss the performance of all three versions of the Ben-Or algorithm discussed in the previous section.

Fair Coin The agent is trained to only control one Byzantine node and there are 10 correct nodes in total. This is because Algorithm 2.3 requires $t < n/10$. The training of the agent lasts for $1,000,000$ steps in total.

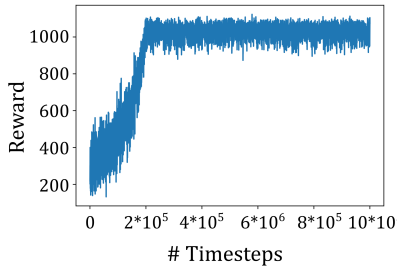
Learning curve: The learning curve (Figure 6.11a) shows a continued learning process of the agent throughout the training. There is no global maximum reward anymore, since the termination of the game is partly stochastic, due to the coin.

Deterministic validation performance: The game is considered to be won, when the algorithm terminates due to $\lfloor n/2 \rfloor + 2t + 1$ correct nodes having received the same coin value. Thereafter, the agent has no influence anymore, since she can only send the same message to all correct nodes. We let the trained agent play 1000 games and she wins all of them. The agent's actions are simple: she works towards an equal number of 0's and 1's in the messages she sends to the correct nodes. If she allowed a majority of seven identical values, no correct node would flip a coin and the algorithm would terminate within the next two steps (see Step 7, Algorithm 2.3).

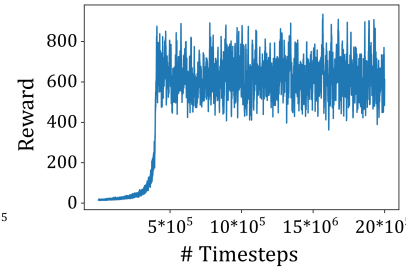
Predefined Coin with Predefined Initial Values In this case, the agent is trained for $2,000,000$ steps, with one Byzantine and 10 correct nodes as in the previous settings.

Learning curve: The learning curve shows a quick and steep learning process (Figure 6.11b). However, the rewards fluctuate a lot after the exploration fraction. As pointed out in Section 6.2.1, the algorithm does not need to terminate at all. If it does, it is either due to the exploration or a mistake by the agent. It is therefore even more crucial to focus on deterministic validation performance.

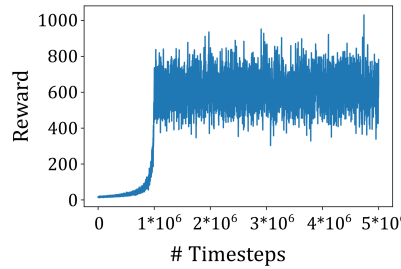
Deterministic validation performance: Because the algorithm can run forever, given that the agent has learned, we impose a maximum number of $10,000$ phases after which we stop the game and consider the agent to have won. The trained agent wins 1000/1000 games. In this case, the agent's actions follow the principle by which we set the initial value, as described in the proof of Lemma 2.4. The agent sends part



(a) Fair coin



(b) Predefined coin



(c) Predefined coin, agent selects initial values

Figure 6.11: Ben-Or algorithm: learning curves smoothed by a moving average of 50 episodes, using code from [113].

of the correct nodes to flip the coin, where their number depends on two aspects:

- The agent cannot allow a majority of more than $\lfloor n/2 \rfloor + 2t$ to form since she would then have no influence in the next round.
- The value of the next coin - if the next coin will have the same value as the current coin, the agent wants the majority not to flip the coin. If the next coin will have the opposite value from the current, the agent lets the majority of the correct nodes flip the coin.

Predefined Coin where the Agent Selects Initial Values In this case, the agent's training consists of $5,000,000$ steps in the setting of 10 correct nodes and one Byzantine node.

Learning curve: The learning curve (Figure 6.11c) looks similar to the previous case. Again, we observe large fluctuations in the reward but want to emphasize the importance of deterministic validation.

Deterministic validation performance: The maximum number of phases is set to $10,000$ and the agent is considered to have won thereafter. Our trained agent wins $1000/1000$ games. It is interesting to study the agent's behavior in the first step of a game, as she develops a similar strategy to the previous case during the rest of the game. For the first step, she selects a vector of initial values, where 4 correct nodes have the value of the first coin, hence, creating a majority for the opposite value to that of the first coin. This is consistent with the initial state described in the proof of Lemma 2.4.

Our results show that deep reinforcement learning is suitable for tackling the asynchronous Byzantine agreement problem. In particular, the networks learn the basic strategies for Byzantine parties that are needed to reach the maximum possible runtime of the Ben-Or algorithm in its different versions. In order to find possible efficient algorithms for solving asynchronous Byzantine agreement using deep reinforcement learning, we also need to train an agent that plays the role of the algorithm in our setting. By letting the algorithm constructor play against the Byzantine agent, one can strive to find efficient consensus algorithms. One possible setting that can be used to implement competing parties is the so-called *self-play*. The implementation of self-play for the asynchronous Byzantine agreement problem will be discussed in the next section.

6.3 Byzantine Agreement with Self-Play

In this section, we discuss the implementation of an asynchronous Byzantine agreement algorithm using self-play. We will use a multi-agent environment with n agents, each representing a node in the system. Since the possible actions of the correct and Byzantine parties in such an algorithm are unbounded, the framework is restricted to model the rounds from blackboard broadcast as described in Section 6.1. We are interested to see whether the correct nodes can learn a policy in a deep reinforcement learning framework to reach agreement and how the Byzantine nodes can prevent them. In this section, we simulate a minimal example with four agents, three of which are correct and one is Byzantine.

In our algorithm, every node broadcasts a message (node ID, round and input value). The Byzantine node can "control" one node per round by hiding its input value in that round. If the Byzantine node is strategic, it

will reveal the hidden value to some correct nodes, such that the local views of the BB-matrix differ. In the following rounds, the Byzantine node can either continue to hide the same node or start hiding some other node while revealing all previously hidden values. In addition to hiding other value, the Byzantine node can also choose its own input value. Since the correct nodes do not advance to the next round of the blackboard matrix before they have received values from 3 nodes in all previous rounds, we can assume that the asynchronous algorithm actually runs in synchronous rounds, where the past views of the nodes can alter over time. This outline corresponds to the properties of the blackboard broadcast according to Definition 6.3.

6.3.1 The Environment

The above setting was implemented in the Python programming language, using the framework Ray [107] for distributed applications and RLlib [77] for deep reinforcement learning as well as constructing the gym-like multi-agent environment. We consider four agents, three of which are correct and have IDs $0, 1, 2$, while the fourth node with ID 3 is Byzantine. The actions, observations, and rewards in this setting are defined as follows:

Action space - correct node this environment maps $\{0, 1\}$ to $\{-1, 1\}$ - the input values that are written to the blackboard matrix and then later observed by the agents. In addition, each correct node has the possibility to lock its action and thus repeatedly write the same value to the blackboard in the following rounds. Locking signals that the correct node is going to terminate its actions in the current round. The two corresponding actions are to terminate with output value -1 or 1. After termination, the value of this node cannot be changed anymore. Observe that this information is not explicitly communicated to the other nodes, instead, other nodes can learn that a node has terminated by observing that its values do not change over the next rounds.

Action space - Byzantine node this environment maps the 64 possible actions of the Byzantine party, $\{0, 1, \ldots, 63\}$, to a list of five elements written as $[-1, 0, 0, 0, 0], [-1, 0, 0, 0, 1], \ldots, [1, 3, 1, 1, 0]$. The first value in the list can be in $\{-1, 1\}$ and it represents the input value of the Byzantine node. The second value is chosen from $\{0, 1, 2, 3\}$, and it determines which node the Byzantine agent decides to hide in the current round, referring to the nodes by their IDs. The next three values model the scheduling and are used as a hide mask. The values are chosen from $\{0, 1\}$, where 0 means that the Byzantine node hides the controlled node value in the current round for the respective node, and 1 means the opposite. Consider for example the action $62 \rightarrow$

$[1, 3, 1, 1, 0]$. This action means that the Byzantine node chooses its input value to be 1, and chooses to hide its own value from the third correct node (*corr_2*).

Observation space - correct node a matrix where the number of rows corresponds to the maximum number of rounds, and the number of columns corresponds to the number of nodes.

Observation space - Byzantine node a matrix where the number of rows is equal to the maximum number of rounds times the number of nodes, and the number of columns is equal to the number of nodes. The Byzantine node is given the observation of a blackboard matrix it sees, as well as all the views of the correct nodes of the up-to-date blackboard matrix.

Termination The environment terminates during the correct nodes' turn if either the maximum number of rounds is reached or all correct nodes want to terminate, either due to All-Same validity or the agreement conditions.

Rewards For each new round that the agents execute, the Byzantine nodes receive a reward of $+1$ and correct nodes receive a penalty of -1. If the correct nodes violate the All-Same validity or the agreement properties, they receive a fixed penalty which is equal to the maximum number of rounds. Note that, due to the termination property, some correct nodes may terminate earlier. In that case, such nodes are assigned the remainder of the reward or penalty after the termination of the environment.

6.3.2 Example of an Environment Rollout

In the beginning, each of the correct nodes is assigned a random input value from $\{-1, 1\}$, these values are added to the first row of the initial blackboard matrix. The Byzantine agent receives the initial values as an observation, renders the agent ready in the *RLlib* setup, and allows the agent to act first. After the Byzantine action, correct nodes receive the observation and are allowed to act.

On a Byzantine agents' turn, the Byzantine agent decides which node to hide, and from which correct nodes to hide the corresponding value. Since some correct nodes may see the hidden value in that round and some may not, the local views of the blackboard matrix for the correct nodes differ (due to scheduling). After the Byzantine action, correct nodes receive their observations - views of the blackboard matrix, where some of the input

values may be hidden, i.e., are 0. Once all correct nodes have received their observations, they are considered ready to play in the environment.

On a correct nodes' turn, each correct agent that has not locked a value yet decides on its input value for the current round. Having chosen the value, the agents write it to the blackboard matrix. Agents that decided to terminate simply write the locked value to the blackboard matrix and do not participate in the learning process anymore. The Byzantine node then receives another observation with these values and the observations of the correct nodes from the previous rounds. This is done to extend the information space of the Byzantine node, such that it has access to the previous rounds and its previous actions, hoping to facilitate its learning process.

Note that the self-play environment allows random actions of the respective agents, and therefore the algorithm of the correct nodes is also randomized. This fact is important since a deterministic strategy would not terminate, as we already discussed in Section 6.2.

6.3.3 Simulation Results

In order to implement self-play, we are using a Proximal Policy Optimization algorithm implemented in the RLlib. The model has two hidden layers of 64 neurons each. With the RLlib multi-agent setup, we have trained both, the correct and Byzantine policy. The results of these initial simulations are presented in the following.

In the experiment, we used an observation of the true blackboard values, as well as access to the previous observations of the correct nodes. The number of iterations for training was set to 200, each iteration containing 8000 timesteps, such that the total number of timesteps in each run was 1600000. The maximum number of rounds for the correct nodes to terminate was 100. In this experiment, the input values for the correct nodes were chosen uniformly at random from the eight possible assignments, such that the probability of all correct nodes to start with the same value was 1/4. To address the possible issues when having randomly-assigned initial values for the correct nodes, we have repeated the simulation multiple times and received similar results.

Figure 6.12 depicts the evolution of the mean policy reward for the nodes, over the course of 200 training iterations. It can be seen that both strategies converge smoothly towards a value close to three rounds.

Figure 6.13 shows the distribution of rounds in 1000 environment test rollouts. Note that this figure only shows the first 20 rounds, as in 99.6% of these tests the correct nodes establish agreement within this time frame. The remaining 0.4% fail to establish agreement on the same value and there-

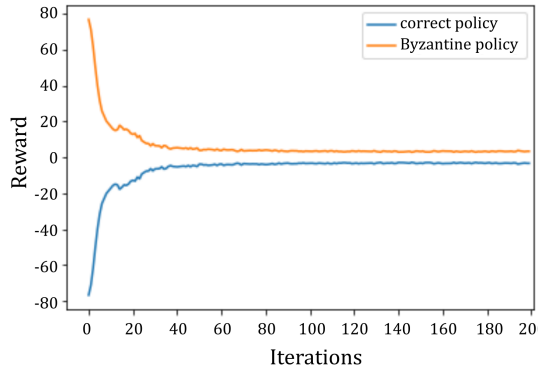


Figure 6.12: Mean policy reward.

fore receive the maximum penalty of 100. Throughout all tests, the All-Same validity was always satisfied and the average episode length was close three rounds.

Byzantine adversary has mainly converged to choosing two actions: the action number 17, which chooses the input value -1 and hides the value of the third correct node from the first and the second node; and action number 37, where the input value of the Byzantine party is 1 and it chooses to hide the value of the first node from the second. Figure 6.14 shows the distribution of Byzantine actions before the first round in which the correct nodes choose their actions. Note that the first actions along the x-axis, i.e. the actions in the left part of the graph, are those where the Byzantine node chooses -1 as its input, while the actions on the right side represent

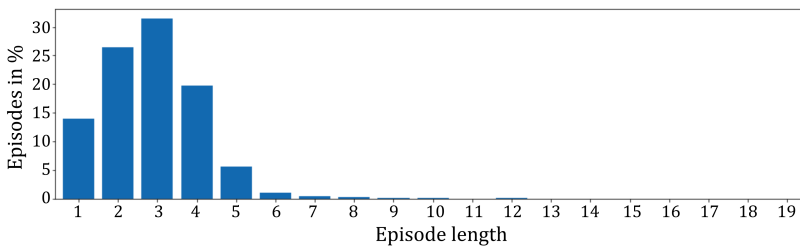
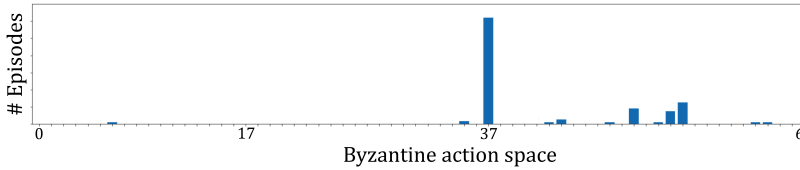
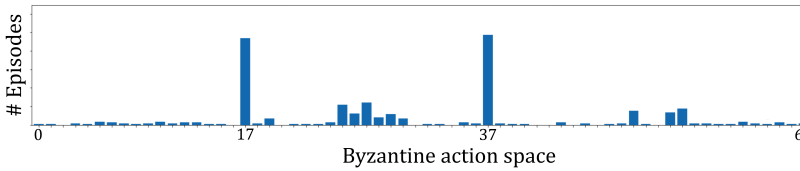


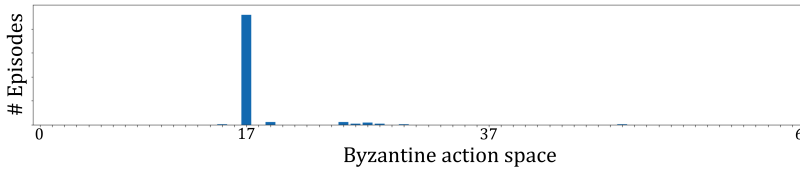
Figure 6.13: Episode length distribution for the first 20 rounds.



(a) Byzantine nodes' actions when the correct nodes start with $[-1, -1, -1]$.



(b) Byzantine nodes' actions when the correct nodes start with $[1, -1, 1]$.



(c) Byzantine nodes' actions when the correct nodes start with $[1, 1, 1]$.

Figure 6.14: Byzantine nodes' actions for different initial input values of correct nodes.

the input value 1. In two of the scenarios, where the correct nodes start out with all same validity, the Byzantine party always chooses the opposite input value. In the scenario, where two of the correct nodes start with 1 and one starts with -1, the Byzantine party learns to choose actions number 17 and 37. The earlier action flips the majority value that one of the correct nodes sees, while the latter action tricks one of the correct nodes to think that All-Same validity is satisfied.

Based on the shown statistics, we can conclude that it is possible to use a self-play scenario in order to simulate Byzantine agreement. We could show that both, the correct and the Byzantine strategies are not exploitable anymore. Another interesting result in this respect is that the strategies of the correct and the Byzantine nodes could be visualized and interpreted in the small setup with four nodes. In particular, we can see the actions that the correct and the Byzantine parties apply in each round, we can relate

them to majority flipping strategies and see that both, the correct and the Byzantine nodes make use of randomness.

6.4 Discussion

In this chapter, we have verified that asynchronous Byzantine agreement can be solved if we assume that the Byzantine nodes are restricted to have power over scheduling but not Byzantine values. This case corresponds to the crash failure model. When Byzantine nodes have power over the scheduling, they can make sure that the correct nodes do not have the same local views. Previous work has mostly focused on Byzantine strategies which are based on different local views of the nodes. The blackboard broadcast that was first proposed in [67] introduces a powerful tool, which can assimilate the nodes' views thus making this initial problem obsolete. While this strategy is sufficient to solve consensus with crash failures, Byzantine scheduling still seems to be problematic. The novel idea to try to detect Byzantine nodes in the non-cryptographic setting seems to be promising. However, as we have shown in Section 6.1, a simple stopping strategy of the controlled nodes can become a problem for detection algorithms.

We therefore investigated the possibility of using deep reinforcement learning in order to see their potential in verifying existing asynchronous Byzantine agreement algorithms or even develop new strategies. In Section 6.2 we have shown that Byzantine behavior can be learned by a reinforcement learning framework. These results provide a foundation for applying self-play in the asynchronous Byzantine agreement setting. Another important aspect of implementing self-play is the blackboard broadcast, as it restricts the action space for Byzantine parties to only 64 actions. The presented preliminary results show that it is significantly harder to learn the Byzantine strategy in a setup, where the correct nodes can exploit randomized actions. While these results are tested on small-scale agreement problems, they are a basis to investigate more complicated Byzantine strategies in the future.

7

Conclusion

In this dissertation, different communication models for the Byzantine agreement problem over public channels have been investigated. In particular, new validity conditions have been introduced for synchronous Byzantine agreement problem, such as the interval validity or the Pareto validity. These validity conditions can be used in a distributed system to agree on a reasonable value. The analysis of the corresponding Byzantine agreement algorithms shows that besides looking for agreement on any value, it is also important to consider the optimization problem of how "close" the agreement value is to the desired result. While it is possible to solve such tasks by agreeing on every input value and then deciding on the approximation value, it could be shown that standard multi-valued synchronous Byzantine agreement algorithms can be extended to achieve these more specific validity conditions. Using such modifications, it could be shown that agreement can be achieved in an almost optimal number of $3(t+1)$ rounds with almost optimal message complexity. In the particular case of rankings, the optimal bound of $t < n/3$ Byzantine nodes could also be achieved. This is not obvious, as the resilience of classic multi-dimensional Byzantine agreement algorithms highly depends on the number of dimensions. Going forward, it would be interesting to investigate to which extent rankings can be applied in approximate agreement in the asynchronous model and how well the re-

sults would approximate the desired ranking, e.g., the Kemeny median. This step is natural, as many proposed multi-dimensional algorithms also solve approximate agreement in the asynchronous communication model. Our results on Byzantine agreement on rankings can also be of interest to social choice theory. There, the goal is to establish agreement on a ranking that represents the intention of the voters. There, the typically considered adversary are selfish voters. Selfish behavior is however not the only behavior that needs to be considered in elections. Voters can also expose random, unpredictable, or even Byzantine behavior. We believe that including Byzantine behavior will help to develop more robust voting rules for elections and we hope that this work can serve as a starting point.

This work has also focused on simplifying the analysis for Blockchain protocols. In particular, a special shared memory model has been presented, that helps to abstract away peer-to-peer communication and thus define boundaries for possibility and impossibility results of Byzantine agreement in blockchain systems. Through the append memory, it could be shown that the DAG structure is more resilient than the chain structure when message delays in the system increase. It was further shown that the DAG structure satisfies optimal resilience and therefore cannot be further improved. We believe that the simplicity of the append memory model improves the understanding of possibility and impossibility results in blockchain protocols and will help to unify their discussion in a single model. It would moreover be interesting to investigate how other blockchain protocols behave in the append memory system and whether their analysis also simplifies significantly through our model.

In addition to the above, this dissertation has also considered the fundamental problem of asynchronous Byzantine agreement with binary input values and the All-Same validity condition. It was shown that complicated Byzantine agreement algorithms open doors to Byzantine strategies for which it becomes even more complicated to prove the correctness of algorithms. In particular, many reliable communication rounds with random values as inputs allowed the Byzantine nodes to "decide" on the input values of the controlled correct nodes. Such strategies call for developing simple algorithms for Byzantine agreement or use new analysis methods to test their correctness. We therefore investigated whether using deep reinforcement learning to simulate Byzantine behavior can help us verify the correctness of algorithms. Our experiments show positive results for small instances of four or five nodes. The results also touch upon the complicated models such as self-play for solving Byzantine agreement. As the tested instances are fairly small, the implemented networks cannot yet be used as a general testing framework. In the future, we suggest to simulate larger instances of

asynchronous Byzantine agreement algorithms and thus possibly find novel simple algorithms for solving this task.

Bibliography

[1] M. Abadi, P. Barham, J. Chen, Z. Chen, A. Davis, J. Dean, M. Devin, S. Ghemawat, G. Irving, M. Isard, M. Kudlur, J. Levenberg, R. Monga, S. Moore, D. G. Murray, B. Steiner, P. Tucker, V. Vasudevan, P. Warden, M. Wicke, Y. Yu, and X. Zheng. TensorFlow: A System for Large-Scale Machine Learning. In *12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 16)*, Savannah, GA, Nov. 2016. USENIX Association.

[2] I. Abraham, Y. Amit, and D. Dolev. Optimal Resilience Asynchronous Approximate Agreement. In *Proceedings of the 8th International Conference on Principles of Distributed Systems*, OPODIS'04, pages 229–239, 2005.

[3] M. K. Aguilera and S. Toueg. A simple bivalency proof that t-resilient consensus requires t+1 rounds. *Information Processing Letters*, 71(3):155 – 158, 1999.

[4] N. Ailon, M. Charikar, and A. Newman. Aggregating Inconsistent Information: Ranking and Clustering. *Journal of the ACM*, 55(5):23:1–23:27, 2008.

[5] D. Alistarh, J. Aspnes, V. King, and J. Saia. Communication-Efficient Randomized Consensus. In *Distributed Computing*, pages 61–75. Springer Berlin Heidelberg, 2014.

[6] P. Almasan, J. Suárez-Varela, A. Badia-Sampera, K. Rusek, P. Barlet-Ros, and A. Cabellos-Aparicio. Deep reinforcement learning meets graph neural networks: exploring a routing optimization use case. *arXiv preprint arXiv:1910.07421*, 2019.

[7] N. Alon, M. Merritt, O. Reingold, G. Taubenfeld, and R. Wright. Tight bounds for shared memory systems accessed by byzantine processes. *Distributed Computing*, 18:99–109, November 2005.

[8] K. J. Arrow. *Social Choice and Individual Values*. CT: Cowles Foundation, New Haven, 1st edition, 1951.

[9] K. J. Arrow. *Social Choice and Individual Values*. John Wiley, New York, 2nd edition, 1963.

[10] J. Aspnes. Lower bounds for distributed coin-flipping and randomized consensus. *J. ACM*, 45(3):415–450, 1998.

[11] J. Aspnes and M. Herlihy. Fast randomized consensus using shared memory. *J. Algorithms*, 11(3):441–461, September 1990.

[12] H. Attiya, A. Bar-Noy, and D. Dolev. Sharing memory robustly in message-passing systems. *J. ACM*, 42(1):124–142, January 1995.

[13] H. Attiya and K. Censor. Tight Bounds for Asynchronous Randomized Consensus. *Journal of the ACM*, 55(5):20:1–20:26, November 2008.

[14] H. Attiya and J. Welch. *Distributed Computing: Fundamentals, Simulations and Advanced Topics*. John Wiley & Sons, Inc., Hoboken, NJ, USA, 2004.

[15] B. Baker, I. Kanitscheider, T. Markov, Y. Wu, G. Powell, B. McGrew, and I. Mordatch. Emergent tool use from multi-agent autocurricula. *arXiv preprint arXiv:1909.07528*, 2019.

[16] T. Bansal, J. Pachocki, S. Sidor, I. Sutskever, and I. Mordatch. Emergent complexity via multi-agent competition. *arXiv preprint arXiv:1710.03748*, 2017.

[17] T. D. Barrett, W. R. Clements, J. N. Foerster, and A. I. Lvovsky. Exploratory combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1909.04063*, 2019.

[18] J. Bartholdi, C. A. Tovey, and M. A. Trick. Voting Schemes for which It Can Be Difficult to Tell Who Won the Election. *Social Choice and Welfare*, 6(2):157–165, 1989.

[19] J. J. Bartholdi, C. A. Tovey, and M. A. Trick. The Computational Difficulty of Manipulating an Election. *Social Choice and Welfare*, 6(3):227–241, 1989.

[20] G. W. Bassett and J. Persky. Robust voting. *Public Choice*, 990(3):299–310, 1999.

[21] M. Bawa, A. Gionis, H. Garcia-Molina, and R. Motwani. The price of validity in dynamic networks. In *Proceedings of the 2004 ACM SIGMOD International Conference on Management of Data*, SIGMOD, June 2004.

[22] I. Bello, H. Pham, Q. V. Le, M. Norouzi, and S. Bengio. Neural combinatorial optimization with reinforcement learning. *arXiv preprint arXiv:1611.09940*, 2016.

[23] M. Ben-Or. Another Advantage of Free Choice: Completely Asynchronous Agreement Protocols. In *Proceedings of the Second Annual ACM Symposium on Principles of Distributed Computing*, PODC '83, pages 27–30, 1983.

[24] P. Berman and J. A. Garay. Asymptotically Optimal Distributed Consensus. In *Automata, Languages and Programming: 16th International Colloquium*, ICALP, 1989.

[25] P. Berman, J. A. Garay, and K. J. Perry. Towards Optimal Distributed Consensus. In *30th Annual Symposium on Foundations of Computer Science*, FOCS, October 1989.

[26] N. Betzler, R. Niedermeier, and G. J. Woeginger. Unweighted coalitional manipulation under the borda rule is np-hard. In *IJCAI*, volume 11, pages 55–60, 2011.

[27] P. Blanchard, E. M. El Mhamdi, R. Guerraoui, and J. Stainer. Machine learning with adversaries: Byzantine tolerant gradient descent. In I. Guyon, U. V. Luxburg, S. Bengio, H. Wallach, R. Fergus, S. Vishwanathan, and R. Garnett, editors, *Advances in Neural Information Processing Systems 30*, pages 119–129. Curran Associates, Inc., 2017.

[28] S. Boyd, N. Parikh, E. Chu, B. Peleato, and J. Eckstein. Distributed Optimization and Statistical Learning via the Alternating Direction Method of Multipliers. *Found. Trends Mach. Learn.*, 3(1).

[29] G. Bracha. Asynchronous Byzantine Agreement Protocols. *Information and Computation*, 75(2):130–143, 1987.

[30] G. Bracha and O. Rachman. Randomized consensus in expected $o(n^2 logn)$ operations. In *Proceedings of the 5th International Workshop on Distributed Algorithms*, WDAG '91, pages 143–150, Berlin, Heidelberg, 1992. Springer-Verlag.

[31] G. Bracha and S. Toueg. Asynchronous Consensus and Broadcast Protocols. *Journal of the ACM*, 32(4):824–840, 1985.

[32] F. Brandt, V. Conitzer, U. Endriss, J. Lang, and A. D. Procaccia. *Handbook of Computational Social Choice*. Cambridge University Press, 1st edition, 2016.

[33] C. Cachin, R. Guerraoui, and L. Rodrigues. *Introduction to Reliable and Secure Distributed Programming.* Springer Publishing Company, Incorporated, 2nd edition, 2014.

[34] C. Cachin, K. Kursawe, and V. Shoup. Random Oracles in Constantinople: Practical Asynchronous Byzantine Agreement Using Cryptography. *Journal of Cryptology*, 18:219–246, 2000.

[35] R. Canetti and T. Rabin. Fast Asynchronous Byzantine Agreement with Optimal Resilience. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Theory of Computing*, STOC '93, pages 42–51, 1993.

[36] S. Chaudhuri. More Choices Allow More Faults: Set Consensus Problems in Totally Asynchronous Systems . *Information and Computation* , 105(1):132 – 158, 1993.

[37] H. Chauhan and V. K. Garg. Democratic Elections in Faulty Distributed Systems. In *Distributed Computing and Networking. ICDCN 2013.*, 2013.

[38] Y. Chen, L. Su, and J. Xu. Distributed statistical machine learning in adversarial settings: Byzantine gradient descent. *arXiv preprint arXiv:1705.05491*, 2017.

[39] B. Chor, S. Goldwasser, S. Micali, and B. Awerbuch. Verifiable secret sharing and achieving simultaneity in the presence of faults. In *26th Annual Symposium on Foundations of Computer Science (sfcs 1985)*, pages 383–395, October 1985.

[40] P. D. Christopher Watkins. Technical note: Q-learning. *Machine Learning*, 8:279–292, 05 1992.

[41] P. J. Courtois, F. Heymans, and D. L. Parnas. Concurrent control with "readers" and "writers". *Communications of the ACM*, 14(10):667–668, October 1971.

[42] J. Davies, G. Katsirelos, N. Narodytska, and T. Walsh. Complexity of and algorithms for borda manipulation. In *AAAI*, volume 11, pages 657–662, 2011.

[43] J. Dean, G. Corrado, R. Monga, K. Chen, M. Devin, M. Mao, M. aurelio Ranzato, A. Senior, P. Tucker, K. Yang, Q. V. Le, and A. Y. Ng. Large Scale Distributed Deep Networks. In F. Pereira, C. J. C. Burges, L. Bottou, and K. Q. Weinberger, editors, *Advances in Neural Information Processing Systems 25*, pages 1223–1231. Curran Associates, Inc., 2012.

[44] P. Diaconis and R. L. Graham. Spearman's Footrule as a Measure of Disarray. *Journal of the Royal Statistical Society. Series B (Methodological)*, 39:262–268, 1977.

[45] E. W. Dijkstra. Solution of a problem in concurrent programming control. *Communications of the ACM*, 8(9), 1965.

[46] B. Doerr, L. A. Goldberg, L. Minder, T. Sauerwald, and C. Scheideler. Stabilizing Consensus with the Power of Two Choices. In *Proceedings of the Twenty-third Annual ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA, 2011.

[47] D. Dolev, C. Dwork, and L. Stockmeyer. On the Minimal Synchronism Needed for Distributed Consensus. *J. ACM*, 34(1):77–97, January 1987.

[48] D. Dolev, N. A. Lynch, S. S. Pinter, E. W. Stark, and W. E. Weihl. Reaching Approximate Agreement in the Presence of Faults. *Journal of the ACM*, 33(3):499–516, 1986.

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

[50] C. Dwork, R. Kumar, M. Naor, and D. Sivakumar. Rank Aggregation Methods for the Web. In *Proceedings of the 10th International Conference on World Wide Web*, WWW '01, pages 613–622, 2001.

[51] A. D. Fekete. Asymptotically optimal algorithms for approximate agreement. *Distributed Computing*, 4(1):9–29, 1990.

[52] M. J. Fischer and N. A. Lynch. A Lower Bound for the Time to Assure Interactive Consistency. *Information Processing Letters*, 14(4):183 – 186, 1982.

[53] M. J. Fischer, N. A. Lynch, and M. S. Paterson. Impossibility of Distributed Consensus with One Faulty Process. *Journal of the ACM*, 32(2):374–382, 1985.

[54] M. Fitzi and M. Hirt. Optimally Efficient Multi-valued Byzantine Agreement. In *Proceedings of the Twenty-fifth Annual ACM Symposium on Principles of Distributed Computing*, PODC, July 2006.

[55] J. Garay, A. Kiayias, and N. Leonardos. The Bitcoin Backbone Protocol: Analysis and Applications. In *Advances in Cryptology - EUROCRYPT 2015*, pages 281–310, 2015.

[56] R. Guerraoui, P. Kuznetsov, M. Monti, M. Pavlovič, and D.-A. Seredinschi. The consensus number of a cryptocurrency. In *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, PODC '19, pages 307–316, 2019.

[57] S. Gupta. A non-consensus based decentralized financial transaction processing model with support for efficient auditing. 2016.

[58] M. Herlihy. Wait-free synchronization. *ACM Trans. Program. Lang. Syst.*, 13(1):124–149, January 1991.

[59] C. Hou, M. Zhou, Y. Ji, P. Daian, F. Tramer, G. Fanti, and A. Juels. Squirrl: Automating attack discovery on blockchain incentive mechanisms with deep reinforcement learning. *arXiv preprint arXiv:1912.01798*, 2019.

[60] C. Hua, S. X. Ding, and Y. A. Shardt. A new method for fault tolerant control through q-learning. *IFAC-PapersOnLine*, 51(24):38 – 45, 2018. 10th IFAC Symposium on Fault Detection, Supervision and Safety for Technical Processes SAFEPROCESS 2018.

[61] J. Huang, M. Patwary, and G. Diamos. Coloring big graphs with alphagozero. *arXiv preprint arXiv:1902.10162*, 2019.

[62] J. G. Kemeny. Mathematics without Numbers. *Daedalus*, 88(4):577–591, 1959.

[63] J. G. Kemeny and J. L. Snell. *Mathematical models in the social sciences*. Introductions to higher mathematics. Blaisdell, Waltham (Mass.), 1962.

[64] M. G. Kendall. A New Measure of Rank Correlation. *Biometrika*, 30(1/2):81–93, 1938.

[65] E. Khalil, H. Dai, Y. Zhang, B. Dilkina, and L. Song. Learning combinatorial optimization algorithms over graphs. In *Advances in Neural Information Processing Systems*, pages 6348–6358, 2017.

[66] L. Kiffer, R. Rajaraman, and a. shelat. A better method to analyze blockchain consistency. In *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*, CCS, pages 729–744, 2018.

[67] V. King and J. Saia. Byzantine Agreement in Polynomial Expected Time. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC '13, 2013.

[68] V. King and J. Saia. Faster Agreement via a Spectral Method for Detecting Malicious Behavior. In *Proceedings of the Twenty-fifth Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '14, 2014.

[69] V. King and J. Saia. Byzantine Agreement in Expected Polynomial Time. *J. ACM*, 63(2), March 2016.

[70] V. King and J. Saia. Correction to byzantine agreement in expected polynomial time, JACM 2016. *CoRR*, abs/1812.10169, 2018.

[71] W. Kool, H. Van Hoof, and M. Welling. Attention, learn to solve routing problems! *arXiv preprint arXiv:1803.08475*, 2018.

[72] L. Lamport, R. Shostak, and M. Pease. The Byzantine Generals Problem. *ACM Transactions on Programming Languages and Systems*, 4(3):382–401, July 1982.

[73] J. Lee, J. Won, and J. Lee. Crowd simulation by deep reinforcement learning. In *Proceedings of Motion, Interaction and Gamesm Limassol, Cyprus, November 8-10*, 2018.

[74] C. Li, P. Li, D. Zhou, W. Xu, F. Long, and A. Yao. Scaling nakamoto consensus to thousands of transactions per second. *arXiv preprint arXiv:1805.03870*, 2018.

[75] M. Li, D. G. Andersen, J. W. Park, A. J. Smola, A. Ahmed, V. Josifovski, J. Long, E. J. Shekita, and B.-Y. Su. Scaling Distributed Machine Learning with the Parameter Server. In *Proceedings of the 11th USENIX Conference on Operating Systems Design and Implementation*, OSDI'14, page 583–598, 2014.

[76] M. Li, D. G. Andersen, A. J. Smola, and K. Yu. Communication Efficient Distributed Machine Learning with the Parameter Server. In *Advances in Neural Information Processing Systems 27*, pages 19–27. Curran Associates, Inc., 2014.

[77] E. Liang, R. Liaw, R. Nishihara, P. Moritz, R. Fox, K. Goldberg, J. E. Gonzalez, M. I. Jordan, and I. Stoica. RLlib: Abstractions for distributed reinforcement learning. In *International Conference on Machine Learning (ICML)*, 2018.

[78] M. C. Loui and H. H. Abu-Amara. Memory requirements for agreement among unreliable asynchronous processes. *Advances in Computing research*, 4(163-183):31, 1987.

[79] D. Malkhi, M. Merritt, M. Reiter, and G. Taubenfeld. Objects Shared by Byzantine Processes. In *Distributed Computing*, pages 345–359, 2000.

[80] K. O. May. A Set of Independent Necessary and Sufficient Conditions for Simple Majority Decision. *Econometrica*, 20(4):680–684, 1952.

[81] D. Melnyk, J. Möller, L. Rakic, O. Richter, and R. Wattenhofer. Asynchronous byzantine agreement with reinforcement learning. unpublished manuscript, 2020.

[82] D. Melnyk, Y. Wang, and R. Wattenhofer. Byzantine Preferential Voting. In *Web and Internet Economics (WINE)*, pages 327–340, 2018.

[83] D. Melnyk, Y. Wang, and R. Wattenhofer. Towards the impossibility of a byzantine shared coin. unpublished manuscript, 2020.

[84] D. Melnyk and R. Wattenhofer. Byzantine Agreement with Interval Validity. In *37th Annual IEEE International Symposium on Reliable Distributed Systems*, SRDS, 2018.

[85] D. Melnyk and R. Wattenhofer. The Append Memory Model: Why BlockDAGs Excel Blockchains. In *32nd ACM Symposium on Parallelism in Algorithms and Architectures (SPAA), Philadelphia, Pennsylvania, USA*, July 2020.

[86] H. Mendes and M. Herlihy. Tight Bounds for Connectivity and Set Agreement in Byzantine Synchronous Systems. In *31st International Symposium on Distributed Computing (DISC 2017)*.

[87] H. Mendes and M. Herlihy. Multidimensional Approximate Agreement in Byzantine Asynchronous Systems. In *Proceedings of the Forty-fifth Annual ACM Symposium on Theory of Computing*, STOC, 2013.

[88] H. Mendes, M. Herlihy, N. Vaidya, and V. K. Garg. Multidimensional agreement in Byzantine systems. *Distributed Computing*, 28(6):423–441, 2015.

[89] E. M. E. Mhamdi, R. Guerraoui, and S. Rouault. The hidden vulnerability of distributed learning in byzantium. *arXiv preprint arXiv:1802.07927*, 2018.

[90] F. B. Mismar and B. L. Evans. Deep q-learning for self-organizing networks fault management and radio performance improvement. *2018 52nd Asilomar Conference on Signals, Systems, and Computers*, Oct 2018.

[91] V. Mnih, K. Kavukcuoglu, D. Silver, A. Graves, I. Antonoglou, D. Wierstra, and M. A. Riedmiller. Playing atari with deep reinforcement learning. *CoRR*, abs/1312.5602, 2013.

[92] A. Mostefaoui, H. Moumen, and M. Raynal. Signature-free Asynchronous Byzantine Consensus with $t < n/3$ and $O(n^2)$ Messages. In *Proceedings of the 2014 ACM Symposium on Principles of Distributed Computing*, PODC '14, pages 2–9, 2014.

[93] S. Nakamoto and A. Bitcoin. A peer-to-peer electronic cash system. 2008.

[94] K. Nakashima, S. Kamiya, K. Ohtsu, K. Yamamoto, T. Nishio, and M. Morikura. Deep reinforcement learning-based channel allocation for wireless lans with graph convolutional networks. *arXiv preprint arXiv:1905.07144*, 2019.

[95] OpenAI. Gym. `https://github.com/openai/gym`, Dec. 2019.

[96] OpenAI. Stable-baselines. `https://github.com/hill-a/stable-baselines`, Dec. 2019.

[97] V. Pareto. *Manuale di Economia Politica con una Introduzione alla Scienza Sociale*. Società Editrice Libraria, 1919.

[98] R. Pass, L. Seeman, and A. Shelat. Analysis of the blockchain protocol in asynchronous networks. In *Advances in Cryptology – EUROCRYPT*, pages 643–673, 2017.

[99] R. Pass and E. Shi. Rethinking large-scale consensus. In *2017 IEEE 30th Computer Security Foundations Symposium (CSF)*, pages 115–129, August 2017.

[100] M. Pease, R. Shostak, and L. Lamport. Reaching Agreement in the Presence of Faults. *Journal of the ACM*, 27(2):228–234, April 1980.

[101] T. Pierrot, G. Ligner, S. E. Reed, O. Sigaud, N. Perrin, A. Laterre, D. Kas, K. Beguir, and N. de Freitas. Learning compositional neural programs with recursive tree search and planning. In *Advances in Neural Information Processing Systems 32: Annual Conference on Neural Information Processing Systems 2019, NeurIPS 2019, 8-14 December 2019, Vancouver, BC, Canada*, pages 14646–14656, 2019.

[102] S. A. Plotkin. Sticky bits and universality of consensus. In *Proceedings of the Eighth Annual ACM Symposium on Principles of Distributed Computing*, PODC, pages 159 – 175, 1989.

[103] R. D. Prisco, D. Malkhi, and M. Reiter. On k-Set Consensus Problems in Asynchronous Systems. *IEEE Transactions on Parallel and Distributed Systems*, 12(1), 2001.

[104] A. D. Procaccia, J. S. Rosenschein, and G. A. Kaminka. On the robustness of preference aggregation in noisy environments. In *Proceedings of the 6th international joint conference on Autonomous agents and multiagent systems*, page 66, 2007.

[105] M. O. Rabin. Randomized byzantine generals. In *24th Annual Symposium on Foundations of Computer Science (sfcs 1983)*, pages 403–409, November 1983.

[106] L. Ren. Analysis of nakamoto consensus. *IACR Cryptology ePrint Archive*, page 943, 2019.

[107] RLlib. Ray. `https://github.com/ray-project/ray`, May 2020.

[108] M. Saks, N. Shavit, and H. Woll. Optimal time randomized consensus – making resilient algorithms fast in practice. In *Proceedings of the Second Annual ACM-SIAM Symposium on Discrete Algorithms*, SODA '91, pages 351–362. Society for Industrial and Applied Mathematics, 1991.

[109] D. Silver, A. Huang, C. J. Maddison, A. Guez, L. Sifre, G. Van Den Driessche, J. Schrittwieser, I. Antonoglou, V. Panneershelvam, M. Lanctot, et al. Mastering the game of go with deep neural networks and tree search. *nature*, 529(7587):484, 2016.

[110] D. Silver, J. Schrittwieser, K. Simonyan, I. Antonoglou, A. Huang, A. Guez, T. Hubert, L. Baker, M. Lai, A. Bolton, et al. Mastering the game of go without human knowledge. *Nature*, 550(7676):354–359, 2017.

[111] Y. Sompolinsky and A. Zohar. Secure High-Rate Transaction Processing in Bitcoin. In *Financial Cryptography and Data Security*, pages 507–527, 2015.

[112] T. Srikanth and S. Toueg. Simulating Authenticated Broadcasts to Derive Simple Fault-Tolerant Algorithms. *Distributed Computing*, 2(2):80–94, June 1987.

[113] Stable-Baselines. `https://colab.research.google.com/drive/1L_IMo6v0a0ALK8nefZm6PqPSy0vZIWBT#scrollTo=mPXYbV39DiCj`, Dec. 2019.

[114] D. Stolz and R. Wattenhofer. Byzantine Agreement with Median Validity. In *19th International Conference on Priniciples of Distributed Systems*, OPODIS, 2015.

[115] R. S. Sutton and A. G. Barto. *Reinforcement Learning: An Introduction*. MIT Press, 1998.

[116] N. Tideman. The Single Transferable Vote. *Journal of Economic Perspectives*, 9(1):27–38, 1995.

[117] S. Toueg. Randomized Byzantine Agreements. In *Proceedings of the Third Annual ACM Symposium on Principles of Distributed Computing*, PODC, August 1984.

[118] L. Tseng. Voting in the presence of byzantine faults. In *2017 IEEE 22nd Pacific Rim International Symposium on Dependable Computing (PRDC)*, January 2017.

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

[120] 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, 2013.

[121] A. van Zuylen and D. P. Williamson. *Deterministic Algorithms for Rank Aggregation and Other Ranking and Clustering Problems*, pages 260–273. Springer Berlin Heidelberg, Berlin, Heidelberg, 2008.

[122] C. J. C. H. Watkins. *Learning from delayed rewards*. PhD thesis, University of Cambridge, England, 1989.

[123] R. Wattenhofer. *Blockchain Science: Distributed Ledger Technology*. CreateSpace Independent Publishing Platform, 2019. third edition.

[124] Z. Xiang and N. H. Vaidya. Relaxed Byzantine Vector Consensus. In *20th International Conference on Principles of Distributed Systems (OPODIS 2016)*.

[125] Z. Xiang and N. H. Vaidya. Brief Announcement: Relaxed Byzantine Vector Consensus. In *Proceedings of the 28th ACM Symposium on Parallelism in Algorithms and Architectures*, SPAA, July 2016.

[126] R. Xu and K. J. Lieberherr. Learning self-game-play agents for combinatorial optimization problems. In *Proceedings of the 18th International Conference on Autonomous Agents and MultiAgent Systems, AAMAS '19, Montreal, QC, Canada, May 13-17, 2019*, pages 2276–2278. International Foundation for Autonomous Agents and Multiagent Systems, 2019.

[127] D. Yin, Y. Chen, K. Ramchandran, and P. Bartlett. Defending against saddle point attack in byzantine-robust distributed learning. *arXiv preprint arXiv:1806.05358*, 2018.

[128] W. Zhang and T. G. Dietterich. Solving combinatorial optimization tasks by reinforcement learning: A general methodology applied to resource-constrained scheduling. *Journal of Artificial Intelligence Reseach*, 1:1–38, 2000.

Index