SAPIENZA
UNIVERSITÀ DI ROMA

DEPARTMENT OF COMPUTER SCIENCE
MASTER DEGREE IN CYBERSECURITY

Consistency of the Sleepy protocol of consensus with Markov chains

DISTRIBUTED SYSTEMS AND CRYPTOGRAPHY

Supervisors:
Prof. D. Venturi
Prof. G. A. Di Luna

Candidate:
Matteo Vicari

Academic Year 2023/2024

Contents

List of Tables

List of Figures

1 Introduction

1.1 Background and motivation

The increasing relevance of distributed systems in deploying applications that require great scalability and high security against malicious actors has brought and is continuing to bring great attention to the study of blockchain technologies, focusing efforts on optimizing and creating consensus protocols suited to the needs of the system. Indeed, consensus protocols are at the core of blockchain technology: they ensure that every node in a distributed network agrees on a common state, even in the presence of adversarial actors. The wide variety of market needs, has led to the creation of multiple consensus protocols, each with its own advantages and disadvantages. As a result, there is a continuing need to carefully evaluate the capabilities, potentials and limitations of each, showing their security with mathematical models.

The aim of this thesis is to test an innovative mathematical framework introduced by Kiffer et al. in [6], which has proven effective for analyzing properties of Proof of Work (PoW)-based consensus protocols. We apply this framework to study a relatively new, non-PoW consensus protocol known as Sleepy. This protocol is designed to operate in environments where nodes may become inactive or fail at any moment, reducing the fraction of active honest nodes and potentially increasing the influence of corrupt nodes controlled by adversaries.

Understanding how the Sleepy protocol performs under these conditions is crucial for improving its security guarantees. By this new mathematical framework, this thesis aims to increase the current understanding of Sleepy's consistency property and verify its resilience against powerful adversaries.

1.2 Objectives and contributions

This thesis conducts an in-depth study of the consistency property of the Sleepy consensus protocol. Specifically, it examines the mathematical conditions necessary for the protocol to satisfy consistency by analyzing the security parameters of the system. After reviewing the classical formal proof proposed in the Sleepy whitepaper [12], we apply a new framework that utilizes Markov chains to improve the estimation of a key concept called convergence opportunity.

The main result of this study is the demonstration that the Sleepy protocol is more resilient to adversarial control than previously established. Specifically, we show that the protocol can withstand a higher fraction of corrupt nodes compared to the original assumptions. In the initial whitepaper, the protocol's security was believed to be compromised beyond a certain threshold of adversarial influence. However, our research shows that the protocol can maintain consistency even with a greater portion of compromised nodes, significantly improving its security for specific choices of security parameters. The following table illustrates the improvements in adversarial tolerance:

Table 1: Key improvements in adversarial tolerance

Blockchain type	**Improvement**
Fast-Paced Blockchain	Up to 10% more adversarial power tolerated
Slow-Paced Blockchain	Percentage-point improvements

Note how for slower paced blockchains, the improvements are smaller but still significant. In fact, for highly populated networks, even minor gains in adversarial resistance can have a meaningful impact on the amount of corrupted node tolerable. In addition to proving consistency, this thesis investigates the most effective adversarial attack against the Sleepy protocol. We model this attack and analyze its probability of success in delaying consensus. This analysis enables us to compute an accurate estimate of the T parameter, which is central to the definition of T-consistency. The T parameter quantifies the number of last blocks added to the chain that cannot be considered finalized yet. Therefore, deriving a value for T which is smaller compared to the original whitepaper one, represents a significant improvement for the security and efficiency of the Sleepy protocol.

1.3 Thesis outline

This paper is structured into five chapters, each focusing on a different aspect of the research. They are structured as follows:

- Chapter 2: Blockchain and consensus protocols foundations
  This chapter introduces the key concepts of blockchains and consensus protocols. It begins with a definition of the blockchain concept and its principal features, followed by a review of the state of the art of some modern consensus algorithms.

- Chapter 3: Sleepy model of consensus
  In this chapter, the Sleepy consensus protocol is formally defined, highlighting its operational principles. We described the protocol's behaviour under static corruptions, providing the necessary elements to understand the analysis we made.

- Chapter 4: Consistency property and proof methods
  This chapter reviews the consistency property of the Sleepy protocol. It begins with a formal definition of the consistency property, followed by a review of the

current conditions that ensure Sleepy's consistency. The chapter also introduces the two main proof methods: the classical random variables method and the new proof approach using Markov chains.

- Chapter 5: Markov method application
  In this chapter, we applied the Markov chain proof technique to the Sleepy protocol. The chapter illustrates how convergence opportunities are computed, and it presents the results of the analysis, showing the protocol's increased resistance to adversarial power.

- Chapter 6: Sleepy best attack analysis
  Finally we modelled and analyzed the most effective attack on the Sleepy protocol. The chapter begins by showing the adversary's capabilities and describing the attack model. We provided a pseudocode for the attack to clearly outline the adversary's steps, enabling an unbiased analysis without the influence of any specific programming language. Then it follows the construction of a Markov chain model to evaluate the probability of success of the attack. The chapter concludes with a solution of the related equations and an analysis of the attack's impact on the protocol's ability to reach consensus.

2 Blockchain and consensus protocols foundations

2.1 What is a blockchain

Following the steps that Pass, Seeman and Shelat have covered in [11], it is useful to define a formalisation of the blockchain that is independent from the consensus protocol used, so as to have a clear foundational structure on which to implement the desired changes and at the same time having a clearly delineated common ground to effectively compare different consensus protocols.

A blockchain is an interactive protocol in which each participant has a local variable called *state* containing a list of messages called *chain*. Participants can communicate with each other over a network and can be visualised as nodes within a graph. Nodes then receive inputs, that can be called records or messages, and try to include them in their chain or that of their neighbours. Generally, the chain is composed by blocks of messages, hence the name *blockchain*. Being in the field of distributed systems, nodes participating in the blockchain can be either honest or malicious, depending on whether they correctly follow the instructions imposed by the protocol or behave arbitrarily by performing actions against other nodes and against the system itself. For the blockchain to be functional and secure, it must respect the following properties:

- *consistency*: with overwhelming probability (in T), at any point, the chains of two honest players can differ only in the last T blocks;
- *chain growth*: with overwhelming probability (in T), at any point in the execution the chain of honest players grows by at least T messages in a specific number of last rounds;
- *chain quality*: with overwhelming probability (in T), for any T consecutive messages in any chain held by some honest player, there is a fraction of messages that were contributed by honest players.

Basically, consistency assures us that all honest nodes have the same view of the message list, chain growth assures us that blocks are added to the chain at least at a certain rate, and chain quality assures us that among these added blocks, at least some originated from honest nodes. These three main requirements must be a necessity for a blockchain to be called such. Obviously these are not the only requirements, we can go into more detail and be more demanding in terms of security parameters. For example, taking the consistency property, this does not ensure that the chain that is considered by everyone to be valid, constantly oscillates between two different chains m_1 and m_2: in even rounds all players possess m_1 while in odd rounds they have m_2. Although it is allowed, this situation could generate quite a few problems in real-life applications of the blockchain (think of a payment validation system), so it is necessary to introduce a consistency property that takes into account the future states of the chain: the *future self consistency* property. With the declared aim of making the blockchain a useful technology for real applications, we require that if a message is added to the chain, it cannot be removed, and that if an honest node wants to add a message to the chain, it will have to appear in it eventually. We will call these two properties *persistency* and *liveness* respectively. It is easy to show that a blockchain that satisfies the properties of consistency, future self consistency, chain growth and chain quality consequently also possesses persistency and liveness and thus allow the blockchain to be a *public ledger*, i.e. a method for achieving consensus in a distributed system.

2.2 Blockchain protocols

A blockchain protocol is a pair of algorithms $(\Pi, \mathcal{C})$ where Π is a stateful algorithm that receives a security parameter λ as inputs and maintains a local state $\texttt{state}$. The algorithm $\mathcal{C}(\lambda, \texttt{state})$ outputs an ordered sequence of "records", or "batches", $\vec{m}$ (e.g. such record may be an ordered sequence of transactions). We call $\mathcal{C}(\lambda, \texttt{state})$ the *record chain* of a player with security parameter λ and local variable $\texttt{state}$; to simplify notation, whenever λ is clear from context we often write $\mathcal{C}(\texttt{state})$ to denote $\mathcal{C}(\lambda, \texttt{state})$. Algorithm Π is parameterized by a *validity* predicate V (denoted by Π^V) that encapsulates the semantic properties (e.g., "no double spending") that a blockchain application aims to achieve. $V(\vec{m})$ returns 1 if and only if the chain $\vec{m}$ is valid for some notion of validity.

2.2.1 Blockchain execution

Consider the execution of a blockchain protocol, formally referred as $(\Pi^V, \mathcal{C})$, that is directed by an environment $\mathcal{Z}(1^\lambda)$ where λ is the security parameter. The environment $\mathcal{Z}$ activates a number of parties $1, 2, \ldots, N$ as either *honest* or *corrupted* parties. Honest parties execute Π on input 1^λ with an empty local state $\texttt{state}$; corrupt parties are controlled by an attacker $\mathcal{A}$ which reads all their inputs/message and sets their outputs/messages to be sent.

- The execution proceeds in rounds that model time steps. In round r, each honest player i receives a message m from the environment $\mathcal{Z}$ and attempts to add it to its chain. Each player potentially receives incoming network messages delivered by $\mathcal{A}$. It may then perform any computation, broadcast a message to all other players and update its local state $\texttt{state}_i$.

- $\mathcal{A}$ is responsible for delivering all messages sent by parties (honest or corrupted) to all other parties. $\mathcal{A}$ cannot modify the content of messages broadcast by honest players, but it may delay or reorder the delivery of a message as long as it eventually delivers all messages. We make this assumption to ensure that the protocol works even in the most adverse possible situation, i.e. the scenario where the adversary has total control of message delivery.

- At any point of the execution, $\mathcal{Z}$ can communicate with the adversary $\mathcal{A}$, access the local state of a player i, make a honest party corrupt and uncorrupt a corrupted player. Corrupting (resp. uncorrupting) a player means that the adversary $\mathcal{A}$ is able (resp. is not able) to access its local state and control its actions.

At this point, we can define $\texttt{EXEC}^{(\Pi^V,\mathcal{C})}(\mathcal{A},\mathcal{Z},\lambda)$ as the random variable denoting the joint view of all protocol participants. Note how the joint view of all nodes uniquely determines the execution.

2.2.2 Admissible environments

We are only considering executions with restricted adversaries and environments; these restrictions will be specified by a predicate $\Gamma(\cdot,\cdot,\cdot,\cdot)$.

Definition 2.1 (Admissible environments)**.** We say that the tuple $(N(\cdot),\rho,\Delta(\cdot),\mathcal{A},\mathcal{Z})$ is Γ admissible w.r.t. $(\Pi^V,\mathcal{C})$ if $\mathcal{A}$ and $\mathcal{Z}$ are non uniform probabilistic polynomial time algorithms, $\Gamma(N(\cdot),\rho,\Delta)=1$ and for every $\lambda\in\mathbf{N}$, every view *view* in the support of $\texttt{EXEC}^{(\Pi^V,\mathcal{C})}(\mathcal{A},\mathcal{Z},\lambda)$, the following holds:

1. $\mathcal{Z}$ activates $N=N(\lambda)$ parties in *view*;
2. $\mathcal{A}$ delays messages by at most $\Delta=\Delta(\lambda)$ rounds;
3. at any round r in *view*, $\mathcal{A}$ controls at most $\rho\cdot N(\lambda)$ nodes;
4. in every round r in *view*, $\mathcal{Z}$ sends local inputs m to an honest player i, iff $V(\mathcal{C}(\texttt{state}_i)||m)=1$.

2.2.3 Random oracles

Random oracles are tools to formalise certain functionalities of blockchain protocols that require an operation outside the network and independent of the network state. In an execution with security parameter λ, we assume all parties have access to a random function $H:\{0,1\}^\star\to\{0,1\}^\lambda$ which they can access through two oracles: $H(x)$ simply outputs the results of the function $H(\cdot)$ on the value x and $H.\texttt{ver}(x,y)$ output 1 iff $H(x)=y$ and 0 otherwise. In any round r, the players (as well as $\mathcal{A}$) may make any number of queries to $H.\texttt{ver}(\cdot,\cdot)$. On the other hand, in each round r, each honest player can make only a single query to $H(\cdot)$, and an adversary $\mathcal{A}$ controlling q parties, can make q sequential queries to $H(\cdot)$, one for each party.

2.3 Consensus algorithms state of art

The consensus algorithm is the core technology of the blockchain, allowing it to delineate which nodes in the network actively participate in the functioning of the system and outlining the actions to be followed to keep the data consistent and secure. Thus, different consensus algorithms can cause radical differences between the blockchains that use them in terms of computational efficiency, security and potential applications. In this chapter, we look at the most widely used consensus algorithms, in order to clarify the current state of the art in this complex world. For this comparison, we will follow the paper by Yusoff et al. [15] and other references taken from the same paper. The first thing to consider is that consensus algorithms can be classified into four types: non-Byzantine based, Byzantine based, DAG based and hybrid. Note that we will refer to consensus algorithms that are able to tolerate Byzantine failures and thus function even in the presence of corrupted nodes that behave hostilely towards the system.

2.3.1 PoW

The Proof of Work concept, originally published by Bonneau et al. in [1] and later used in the construction of the Bitcoin blockchain, is perhaps the most popular consensus algorithm. It relies on solving computationally difficult cryptographic problems to determine who has the right to add a block to the chain. This process of solving the crypto puzzle is called mining. Although solving the problem requires energy and time-consuming brute force, verifying the solution is very fast, which makes the blockchain difficult to alter but easy to validate. This ensures the security of the system by preventing an adversary from altering past blocks or adding more blocks for malicious purposes. Each block of transactions is linked to the previous block using a cryptographic hash value. The mining process is performed by choosing a random value called *nonce* and calculating the block header digest. The header contains the nonce and other information that uniquely identifies the content of the block such as the previous block link and the transactions within it. If the digest is less than a target value chosen by the protocol, the block is considered valid and can be sent to all other nodes via broadcast and added to the blockchain. Changing the target value, changes the difficulty of the problem and affects the speed of block creation, increasing or decreasing the rate at which blocks are added to the main chain. The PoW protocol has proven to be a secure consensus algorithm, resistant to the most common network consistency attacks such as double spending or system saturation attempts such as denial of services attacks. It does, however, have limitations due to the high energy requirements needed to solve crypto puzzles and to a relatively low rate of transactions per minute that can be tolerated for the current users needs.

2.3.2 PoS

The Proof of Stake consensus algorithm removes the computational element typical of PoW and introduces the new concept of *stake* (from which it takes its name). Stake is the amount of digital currency that consensus participants must hold in order to propose and validate a block. There are several blockchains that implement consensus through the PoS algorithm and each protocol has its own characteristics. Generally speaking, what happens is that time is divided into precise rounds in which validators are randomly elected from the group of eligible participants; among these validators, some will also have the power to build a block, collecting transactions from a pool, and propose it to the other validators. They will add their signature alongside the one of the block creator. The set of validators must have placed a specific quantity of cryptocurrency locked on a specific wallet of the network. This amount, called stake, becomes a collateral of the block and thus can be seen as the block score. In order to choose the chain to which further blocks should be added, an algorithm calculates the total amount of stake on the chain by simply summing the stake of each block and finally the chain with the highest stake is cosidered the valid one. Security is guaranteed by the fact that all validators, chosen at random, sign the proposed blocks only after checking that the transactions chosen and entered in the list are valid. Each block obviously contains the link to the previous block, and there is no possibility of altering its content due to the hash of the block and the numerous signatures it contains. In addition, validators will receive punishments if they misbehave (e.g. vote for a block in an invalid chain). The classic punishment is slashing, i.e. the removal of a part of the staked cryptocurrency causing an economic loss for misbehaving validators and a decrease in their voting power. The main advantages of PoS are energy efficiency and a higher transaction rate per minute. In fact, the absence of complex computations allows for greater scalability and energy savings compared to PoW. However, it is vulnerable to 51% attack: if an actor accumulates more than 50% of the stake, this would give it control over the network, allowing it to manipulate transactions and blocks. In addition, other possible attacks on the PoS such as the *nothing at stake* attack and the *long range* attack must be considered when constructing the protocol.

2.3.3 PoL

The Proof of Luck algorithm has a simple but effective approach to achieving consensus with the aim of solving the various problems present in previous algorithms. PoL makes use of a trusted execution environment (TEE) platform that allows the protocol to generate a trusted random number used to determine which player is entitled to add the block to the chain. It is therefore a leader election between the nodes of the network and is secure due to the presence of the TEE which prevents an adversary from gaining an unfair advantage and manipulating the blockchain unless they control most of the nodes in the network. The ways to implement the protocol are numerous but they generally have the following characteristics: the algorithm proceeds by rounds and exploits two main functionalities, `PoLRound` and `PoLMine`. `PoLRound` is called at the start of each round by all the nodes participating in the consensus passing as input the block they consider to be the most recent. When the round time expires, each participant invokes the `PoLMine` function which allows them to create a new block and attach it to the previous one via the link in the header. At this point, `PoLMine` generates a random value between $[0, 1]$ with a uniform distribution that allows it to determine the new winning block from among the several proposed in this round. The procedure is explained in more detail by Milutinovic et al. in [8]. The Sleepy protocol of consensus that we will discuss later is based on this very method of consensus. PoL, due to its strong reliance on luck, possesses a high immunity to the most common blockchain attacks and allows a very high speed in reaching consensus with minimal energy usage. However, the main problems associated with thess type of protocols are the high trust placed in the TEE (which for some could undermine the principle of decentralisation) and the necessary synchronisation of the system. Indeed, having the network clocks synchronised is an essential requirement for nodes to be able to make function calls in the proper instants.

2.3.4 PoB

Proof of Burn is a special algorithm that allows users of a blockchain to burn digital currency. It can be used for various purposes, for instance to decrease the amount of a cryptocurrency or to implement a consensus protocol. Being a relatively new and unusual algorithm, so far it had no particular application and there is limited awareness of its potential and limitations. You can consult a precise formalisation in [5]. The main idea behind this algorithm is the destruction of cryptocurrency through a transaction to a dedicated wallet. The wallet is purposely generated by the system as a storage for the burnt cryptocurrency and has no private key, which guarantees that the digital assets it holds can never be reused. Depending on how the system is implemented, it may also be possible to burn cryptocurrency from other blockchains. The purpose of burning is to show a long-term commitment to the system by giving up an immediate gain for a potential future profit. Decreasing the amount of cryptocurrency in circulation generates a deflationary process that allows an increase in value of the burnt currency. In addition, since in PoB the burn action is verifiable by everyone through a special functionality that checks the transaction signature and that the recipient wallet is a valid burn wallet, it can be used as a criterion to determine who is entitled to publish the next block. The greater the amount of currency burnt by a miner, the greater his chances of being selected to mine the next block.

2.3.5 Hybrids and Variations

There are consensus protocols that possess the structure and basis of the main protocols listed above but implement variations to make them more secure or efficient. One of these is the delayed Proof of Work (dPoW) in which in addition to the main blockchain called the primary chain, a secondary chain called the notary chain is added. In this protocol, when a miner manages to mine a block and add it to the primary chain, he must add its hash to the notary chain. In this way, the two chains grow in parallel and the notary chain can be used to verify the validity of the primary chain.

In a similar way, the PoS protocol was also altered with the introduction of the Delegated Proof of Stake (DPoS). In this consensus protocol, stake holders participate in a voting process that leads to the election of a set of validators called delegates who have the power to create the block and validate it. Adding this distributed voting process provides a layer of protection to the 51% attack.

There are even consensus algorithms that merge the current main protocols by implementing a system that utilises both PoW and PoS elements simultaneously. The main mechanisms can in fact be combined in sequence or in parallel with the aim of obtaining the advantages of the protocols used while improving their efficiency and reducing the impact of their structural flaws.

3 Sleepy model of consensus

3.1 Definitions and protocol description

We will try to apply the new method of calculating convergence opportunities proposed by Kiffer, Rajaraman and Shelat in [6] on a distributed protocol in a “sleepy” model of computation, where players can be either online (alert) or offline (asleep), and their online status may change at any point during the protocol. We chose this protocol because we realised that an estimation performed in Sleepy’s reference paper [12] would benefit from this new method of analysis, and Kiffer et al., in their paper [6], mention it as a possible protocol in which to apply their new analysis. The Sleepy protocol aims to remove the PoW, while still maintaining many of its basic ideas, in a weakly synchronised network. All descriptive information and definitions of the model have been taken from the Sleepy reference paper [12] and have been quoted in order to contextualise the upcoming results and facilitate the reading of the notation and terminology. For more details about the Sleepy protocol, please refer directly to its official paper mentioned above.

3.1.1 (Weakly) synchronized clocks

Sleepy allows nodes clocks to be offset by a bounded amount, commonly referred to as weakly synchronized clocks. It is possible to apply a general transformation such that we can translate the clock offset into the network delay, and consequently in the formal model we may simply assume that nodes have synchronized clocks without loss of generality. Therefore henceforth in this paper we consider a model with a globally synchronized clocks (without losing the ability to express weak synchrony). Each clock tick is referred to as an atomic time step. Nodes can perform unbounded polynomial amount of computation in each atomic time step, as well as send and receive polynomially many messages.

3.1.2 Public-key infrastructure

We assume the existence of a public-key infrastructure (PKI). Specifically, we shall assume that the PKI is an ideal functionality $\mathcal{F}_{CA}$ (available only to the present protocol instance) that does the following:

- On receive `register`(upk) from $\mathcal{P}$: remember $(upk, \mathcal{P})$ and ignore any future message from $\mathcal{P}$;
- On receive `lookup`($\mathcal{P}$): return the stored upk corresponding to $\mathcal{P}$ or $\perp$ if none is found.

3.1.3 Corruption model

At the beginning of any time step t, the environment $\mathcal{Z}$ can issue instructions of the form

$$(\texttt{corrupt}, i) \text{ or } (\texttt{sleep}, i, t_0, t_1) \text{ where } t_1 \geq t_0 \geq t$$

$(\texttt{corrupt}, i)$ causes node i to become corrupt at current time, whereas $(\texttt{sleep}, i, t_0, t_1)$ where $t_1 \geq t_0 \geq t$ will cause node i to sleep during $[t_0, t_1]$. A node can be in one of the following states:

- *Honest.* An honest node can either be awake or asleep (or sleeping/sleepy). Henceforth we say that a node is alert if it is honest and awake. When we say that a node is asleep (or sleeping/sleepy), it means that the node is honest and asleep.
- *Corrupt.* Without loss of generality, we assume that all corrupt nodes are awake.

Henceforth, we say that corruption (or sleepiness resp.) is static if $\mathcal{Z}$ must issue all `corrupt` (or `sleep` resp.) instructions before the protocol execution starts. We say that corruption (or sleepiness resp.) is adaptive if $\mathcal{Z}$ can issue `corrupt` (or `sleep` resp.) instructions at any time during the protocol's execution.

3.1.4 Network delivery

The adversary is responsible for delivering messages between nodes. We assume that the adversary $\mathcal{A}$ can delay or reorder messages arbitrarily, as long as it respects the constraint that all messages sent from honest nodes must be received by all honest nodes in at most Δ time steps.

3.1.5 Compliant executions

We use the notation $\mathit{view} \leftarrow_{\$} \texttt{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \lambda)$ to denote a randomized execution of the protocol Π with security parameter λ and w.r.t. to an $(\mathcal{A}, \mathcal{Z})$ pair. Specifically, *view* is a random variable containing an ordered sequence of all inputs, outputs, and messages sent and received by all Turing Machines during the protocol's execution. We use the notation $|\mathit{view}|$ to denote the number of time steps in the execution trace *view*. Globally, we will use N to denote (an upper bound on) the total number of nodes, and N_{crupt} to denote (an upper bound on) the number of corrupt nodes, and Δ to denote the maximum delay of messages between alert nodes. Then we impose that if an alert node i gossips a message at time t in *view*, then any node j alert at time $t' \geq t + \Delta$ (including ones that wake up after t) will have received the message.

3.2 Sleepy consensus under Static Corruptions

In this section, we will describe our basic Sleepy consensus protocol that is secure under static corruptions and static sleepiness. In other words, the adversary (and the environment) must declare upfront which nodes are corrupt as well as which nodes will go to sleep during which intervals. Furthermore, the adversary (and the environment) must respect the constraint that at any moment of time, roughly speaking the majority of online nodes are honest. For simplicity, we will describe our scheme pretending that there is a random oracle H. We assume that the random oracle H instance is not shared with other protocols, and that the environment $\mathcal{Z}$ is not allowed to query the random oracle H directly, although it can query the oracle indirectly through $\mathcal{A}$.

3.2.1 Valid blockchain

A blockchain is a chain of blocks. Let's define a valid block and a valid blockchain for this protocol. We say that a tuple

$$B := (\hbar_{-1}, \text{txs}, \text{time}, \mathcal{P}, \sigma, \hbar)$$

is a valid block iff

- $\Sigma.\texttt{ver}_{pk}((\hbar_{-1}, \text{txs}, \text{time}); \sigma) = 1$ where $pk := \mathcal{F}_{CA}.\texttt{lookup}(\mathcal{P})$; and
- $\hbar = d(\hbar_{-1}, \text{txs}, \text{time}, \mathcal{P}, \sigma)$, where $d : \{0,1\}^* \to \{0,1\}^\lambda$ is a collision resistant hash function.

Now let $\texttt{eligible}^t(\mathcal{P})$ be a function that determines whether a party $\mathcal{P}$ is an eligible leader for time step t. Let chain denote an ordered chain of real-world blocks, we say that chain is a valid blockchain w.r.t. $\texttt{eligible}$ and time t iff

- $\mathit{chain}[0] = \mathit{genesis} = (\bot, \bot, \text{time} = 0, \bot, \bot, \hbar = 0)$, commonly referred to as the genesis block;
- $\mathit{chain}[-1].\text{time} \le t$; and
- for all $i \in [1, \ell]$ where $\ell := |\mathit{chain}|$, the following holds:
  1. $\mathit{chain}[i]$ is a valid block;
  2. $\mathit{chain}[i].\hbar_{-1} = \mathit{chain}[i-1].\hbar$;
  3. $\mathit{chain}[i].\text{time} > \mathit{chain}[i-1].\text{time}$, i.e. block-times are strictly increasing;
  4. let $t := \mathit{chain}[i].\text{time}$, $\mathcal{P} := \mathit{chain}[i].\mathcal{P}$, it holds that $\texttt{eligible}^t(\mathcal{P}) = 1$.

3.2.2 Protocol functioning

The protocol takes a parameter p as input, where p corresponds to the probability each node is elected leader in a single time step. All nodes that just spawned will invoke the `init` entry point. During initialization, a node generates a signature key pair and registers the public key with the public-key infrastructure $\mathcal{F}_{CA}$. Now, the basic Sleepy protocol proceeds very much like a proof-of-work blockchain, except that instead of solving computational puzzles, a node can extend the chain at time t iff it is elected leader at time t. To extend the chain with a block, a leader of time t simply signs a tuple containing the previous block's hash, the node's own party identifier, the current time t, as well as a set of transactions to be confirmed. Leader election can be achieved through a public hash function H that is modeled as a random oracle. Although the scheme is described assuming a random oracle H, it is not hard to observe that it can be replaced with a common reference string `crs` and a pseudo-random function `PRF`. Specifically, the common reference string $k_0 \leftarrow_\$ 0, 1^\lambda$ is randomly generated after $\mathcal{Z}$ spawns all corrupt nodes and commits to when each honest node shall sleep. Then, we can simply replace calls to $H(\cdot)$ with $\texttt{PRF}_{k_0}(\cdot)$.

Algorithm 1 Protocol $\Pi_{\text{sleepy}}(p)$

```
On input init() from $\mathcal{Z}$:
    let $(pk, sk) := \mathfrak{s}.\texttt{gen}()$, register $pk$ with $\mathcal{F}_{CA}$, let $chain := genesis$

On receive $chain'$:
    assert $|chain'| > |chain|$ and $chain'$ is valid w.r.t. eligible and time $t$;
    $chain := chain'$ and gossip $chain$

Every time step:
•   receive input transactions(txs) from $\mathcal{Z}$
•   let $t$ be the current time, if $\texttt{eligible}^t(\mathcal{P})$ where $\mathcal{P}$ is the current node identifier:
        let $\mathfrak{s} := \Sigma.\texttt{sign}(sk, chain[-1].\hbar, \text{txs}, t)$, $\hbar' := d(chain[-1].\hbar, \text{txs}, t, \mathcal{P}, \mathfrak{s})$,
        let $B := (chain[-1].\hbar, \text{txs}, t, \mathcal{P}, \mathfrak{s}, \hbar')$, let $chain := chain \| B$ and gossip $chain$
•   output extract($chain$) to $\mathcal{Z}$ where extract outputs an ordered list of txs

Subroutine $\text{eligible}^t(\mathcal{P})$:
    return 1 if $H(\mathcal{P}, t) < D_p$ and $\mathcal{P}$ is a valid party of this protocol; else return 0
```

The difficulty parameter D_p is defined such that the hash outcome is less than D_p with probability p. Note that the protocol $\Pi_{\text{sleepy}}(p)$ is parametrized in p, that is the probability that any node is elected leader in any time step. Looking ahead, due to some compliance rules, it is sufficient for the protocol to have foreknowledge of N, Δ, the maximum clock offset of the system, and c, the expected blocktime, since we will define p as the function of these parameters.

Since proving results may be too complicated in $\Pi_{\text{sleepy}}(p)$, lets start by defining a simplification that we will denote by Π_{ideal}. In this ideal protocol, nodes interact with an ideal functionality denoted $\mathcal{F}_{\text{tree}}(p)$, which keeps track of all valid chains at each time instant. In the official Sleepy paper, it is shown that the real-world Sleepy protocol is emulated perfectly by its ideal counterpart, and at the same time all demonstrations made in the ideal are valid in the real world protocol.

3.2.3 Simplified ideal protocol

Algorithm 2 $\mathcal{F}_{\text{tree}}(p)$

```
On init: $tree := genesis$, $\texttt{time}(genesis) := 0$

On receive $\texttt{leader}(\mathcal{P}, t)$ from $\mathcal{A}$ or internally:
    if $\Gamma[\mathcal{P}, t]$ has not been set, let $\Gamma[\mathcal{P}, t] := \begin{cases} 1 & \text{with probability } p \\ 0 & \text{o.w.} \end{cases}$
    return $\Gamma[\mathcal{P}, t]$

On receive $\texttt{extend}(chain, B)$ from $\mathcal{P}$: let $t$ be the current time:
    assert $chain \in tree$, $chain \| B \notin tree$, and $\texttt{leader}(\mathcal{P}, t)$ output 1
        append $B$ to $chain$ in $tree$, record $\texttt{time}(chain \| B) := t$, and return $succ$

On receive $\texttt{extend}(chain, B, t')$ from corrupt party $\mathcal{P}^*$: let $t$ be the current time
    assert $chain \in tree$, $chain \| B \notin tree$, $\texttt{leader}(\mathcal{P}^*, t')$ output 1
        assert $\texttt{time}(chain) < t' \leq t$
            append $B$ to $chain$ in $tree$, record $\texttt{time}(chain \| B) := t'$, and return $succ$

On receive $\texttt{verify}(chain)$ from $\mathcal{P}$: return $(chain \in tree)$
```

$\mathcal{F}_{\text{tree}}(p)$ flips random coins to decide whether a node is the elected leader for every time step, and an adversary $\mathcal{A}$ can query this information (i.e., whether any node is a leader in any time step) through the `leader` query interface. Finally, alert and corrupt nodes can call $\mathcal{F}_{\text{tree}}(p)$.`extend` to extend known chains with new blocks. $\mathcal{F}_{\text{tree}}(p)$ will then check if the caller is a leader for the time step to decide if the extend operation is allowed. $\mathcal{F}_{\text{tree}}(p)$ keeps track of all valid chains, such that alert nodes will call $\mathcal{F}_{\text{tree}}(p)$.`verify` to decide if any chain they receive is valid. Alert nodes always store the longest valid chains they have received, and try to extend it. Observe that $\mathcal{F}_{\text{tree}}(p)$

Algorithm 3 Protocol Π_{ideal}

```
On init: $chain := genesis$

On receive $chain'$:
  if $|chain'| > |chain|$ and $\mathcal{F}_{tree}.\texttt{verify}(chain') = 1$: $chain := chain'$, gossip $chain$

Every time step:
  • receive input $B$ from $\mathcal{Z}$
  • if $\mathcal{F}_{tree}.\texttt{extend}(chain, B)$ outputs $succ$: $chain := chain \| B$ and gossip $chain$
  • output $chain$ to $\mathcal{Z}$
```

has two entry points named `extend`, one of them is the honest version and the other is the corrupt version. In this ideal protocol, alert nodes always mine in the present, i.e., they always call the honest version of extend that uses the current time t. In this case, if the honest node succeeds in mining a new chain denoted *chain*, $\mathcal{F}_{\text{tree}}(p)$ records the current time t as the chain's block-time by setting $\mathcal{F}_{\text{tree}}(p)(view).\texttt{time}(chain) = t$. On the other hand, corrupt nodes are allowed to call a malicious version of `extend` and supply a past time step t_0. When receiving an input from the adversarial version of `extend`, $\mathcal{F}_{\text{tree}}(p)$ verifies that the new block's purported time t_0 respects the strictly increasing rule. If the corrupt node succeeds in mining a new block, then $\mathcal{F}_{\text{tree}}(p)$ records the purported time t_0 as the chain's block-time. Note that if an alert node mines a chain at time t, then the chain's block-time must be t as well. By contrast, if a corrupt node mines a chain at time t, the chain's block-time may not be truthful, in fact it may be smaller than t.

3.2.4 Useful notation

To make reading easier, we collected in table 2 the most important notations and parameters that we have encountered and will encounter in the course of the paper. We have tried to keep the notation as close as possible to the main reference papers (i.e. [6], [12], [11]) at least in the common parts. Where there has been a need to define new notations given the inevitable differences in the consensus protocol under consideration and the particular type of analysis, these have been precisely defined before being used. This will occur mainly in the chapters 5 and 6.

Table 2: Notation and parameters

Symbol	**Definition**	**Formula**
$\mathcal{A}$	Adversarial entity	
Δ	Maximum network delay	
c	Average round-time to win the leader election	
N	Total of nodes in the system	$N_{alert} + N_{sleepy} + N_{crupt}$
p	Probability of winning the leader election	$\frac{1}{c \cdot \Delta}$
μ	Fraction of honest alert nodes	$\frac{N_{alert}}{N}$
σ	Fraction of honest asleep nodes	$\frac{N_{sleepy}}{N}$
ρ	Fraction of corrupted nodes	$\frac{N_{crupt}}{N}$
α	Expected honest leaders per round	$p \cdot N_{alert}$
β	Expected corrupt leaders per round	$p \cdot N_{crupt}$

4 Consistency property and proof methods

4.1 Consistency property definition

Roughly speaking, consistency stipulates common prefix and future self-consistency. Common prefix requires that all honest nodes' chains, except for roughly $O(\lambda)$ number of trailing blocks that have not stabilized, must all agree. Future self-consistency requires that an honest node's present chain, except for roughly $O(\lambda)$ number of trailing blocks that have not stabilized, should persist into its own future. These properties can be unified in the following formal definition (which additionally requires that at any time, two alert nodes' chains must be of similar length).

Let $\text{consistent}^T(\mathit{view}) = 1$ iff for all times $t \leq t'$, and all players i, j (potentially the same) such that i is alert at t and so is j at t' in *view*, we have that the prefixes of $\mathit{chain}_i^t(\mathit{view})$ and $\mathit{chain}_j^{t'}(\mathit{view})$ consisting of the first $\ell = |\mathit{chain}_i^t(\mathit{view})| - T$ records are identical, this also implies that the following must be true: $\mathit{chain}_j^{t'}(\mathit{view}) > \ell$, i.e., $\mathit{chain}_j^{t'}(\mathit{view})$ cannot be too much shorter than $\mathit{chain}_i^t(\mathit{view})$ given that $t' \geq t$.

Definition 4.1 (Consistency)**.** A blockchain protocol Π satisfies T_0-*consistency*, if for all Π-compliant pair $(\mathcal{A}, \mathcal{Z})$, there exists some negligible function `negl` such that for every sufficiently large $\lambda \in \mathbf{N}$ and every $T \geq T_0$ the following holds:

$$Pr\left[\mathit{view} \leftarrow_{\$} \texttt{EXEC}^{\Pi}(\mathcal{A}, \mathcal{Z}, \lambda) : \text{consistent}^T(\mathit{view}) = 1\right] \geq 1 - \texttt{negl}(\lambda)$$

Additionally, we say that a blockchain protocol Π satisfies T_0 *-consistency* w.r.t. failure probability $\texttt{negl}(\cdot)$ if the above definition is satisfied when the negligible function is fixed to $\texttt{negl}(\cdot)$ for any Π-*compliant* $(\mathcal{A}, \mathcal{Z})$.

Note that a direct consequence of consistency is that at any time, the chain lengths of any two alert players can differ by at most T (except with negligible probability).

4.1.1 Convergence Opportunities

With the aim of studying the consistency property of PoW consensus protocols, Pass et al. in [11] consider any window of T rounds and count special events, called *Convergence Opportunities*, which are events after which all honest players agree on a single chain; we will define them formally shortly. If an adversary wants to break consistency, they must combat all convergence opportunities. It follows that counting how many convergence opportunities are present in a generic sequence of events in the blockchain is crucial to study the consistency of the protocol under consideration. Obviously, the definition of the concept of convergence opportunity varies slightly depending on the protocol considered, although it maintains the same basic structure and idea.

In the Sleepy protocol, a convergence opportunity is a Δ-period of silence in which no alert node is elected leader, followed by a time step in which a single alert node is elected leader, followed by another Δ-period of silence in which no alert node is elected leader. More formally:

Definition 4.2 (Convergence Opportunity)**.** Given a *view*, suppose $T \leq |view| - \Delta$, we say that $[T - \Delta, T + \Delta]$ is a convergence opportunity iff

- $\forall\, t \in [max(0, T - \Delta), \Delta)$, no alert node at time t is elected leader;
- A single alert node is elected leader at time T;
- $\forall\, t \in (T, T + \Delta]$, no alert node at time t is elected leader.

Let T denote the time in which a single alert node is elected leader during a convergence opportunity. For convenience, we often use T to refer to the convergence opportunity. We say that a convergence opportunity T is contained within a window $[t' : t]$ if $T \in [t' : t]$. Henceforth, we use the notation $\mathbf{C}(view)[t' : t]$ to denote the number of convergence opportunities contained within the window $[t' : t]$ in *view*.

Given the above definition, it is evident how a convergence opportunity leads to the system's consensus on the blocks that form the main chain: assuming we start from a messy state in which some nodes in the system have not received the most recent blocks, after an initial interval of Δ rounds without elections, the latter will have received the missing blocks, since Δ corresponds to the maximum delay in the network. At this point, the next elected leader will propose a new block, which will reach all alert nodes within Δ consecutive rounds without leaders.

4.2 Current Sleepy consistency conditions

In this section, we list the main theoretical results that lead to the determination of consistency conditions for Sleepy presented by Pass et al. in [12]: these conditions are basically equations on the parameters of the protocol whose solutions guarantee that the system respects the consistency property.

The first essential step is to estimate the number of convergence opportunities in a time interval of the execution of the consensus algorithm.

Lemma 1 (Number of convergence opportunities for any fixed window)**.** For any $t_0, t_1 \geq 0$ such that $t := t_1 - t_0 > 0$, any $\Pi_{ideal} -$ *compliant pair* $(\mathcal{A}, \mathcal{Z})$, for any positive constant η, there exists a constant η', such that for any $\lambda \in \mathbb{N}$, the following holds:

$$Pr\left[view \leftarrow_{\$} \texttt{EXEC}^{\Pi_{ideal}}(\mathcal{A}, \mathcal{Z}, \lambda) : \mathbf{C}(view)[t_0, t_1] \leq (1-\eta)(1-2pN\Delta)\alpha t\right] < e^{-\eta'\alpha t}$$

Basically, we can say that with a good probability of success, alert nodes will have at least $(1-2pN\Delta)\alpha t$ convergence opportunities available in the time interval of length t.

At this point, it is necessary to quantify the maximum rounds available to the adversary to disrupt convergence opportunities. In other words, it is useful to know how many time slots the adversary has available with a corrupted leader at its disposal, which allows it to propose malicious blocks to only part of the network, delaying the delivery to the others and undermining the consensus of the system.

Given a *view*, let $\mathbf{A}(view)[t_0 : t_1]$ denote the number of time steps in which at least one corrupt node is elected leader during the window $[t_0 : t_1]$.

Lemma 2 (Upper bound on adversarial time slots for any fixed window)**.** For any t_0, t_1 such that $t := t_1 - t_0 \geq 0$, for any Π_{ideal} – *compliant pair* $(\mathcal{A}, \mathcal{Z})$, for any constant $0 < \epsilon < 1$ and any $\lambda \in \mathbb{N}$,

$$Pr\left[view \leftarrow_{\$} \texttt{EXEC}^{\Pi_{ideal}}(\mathcal{A}, \mathcal{Z}, \lambda) : \mathbf{A}(view)[t_0, t_1] > (1+\epsilon)\beta t\right] \leq e^{-\frac{\epsilon^2 \beta t}{3}}$$

This means that, with a good probability of success, corrupted nodes will have at most βt adversarial time slots available.

Finally, we can compare the two quantities and study for which values of the security parameters the convergence opportunities are greater than the adversarial time slots. This ensures us that, although the adversary may succeed in slowing down the process of reaching consensus, it will never succeed in disrupting all convergence opportunities, and the alert nodes will succeed in maintaining a common prefix.

Lemma 3 (Adversarial time slots vs convergence opportunities for any fixed window)**.** For any t_0, t_1 such that $t := t_1 - t_0 \geq 0$, for any Π_{ideal} – *compliant pair* $(\mathcal{A}, \mathcal{Z})$, there exists some positive constant η, such that for any $\lambda \in \mathbb{N}$,

$$Pr\left[view \leftarrow_{\$} \texttt{EXEC}^{\Pi_{ideal}}(\mathcal{A}, \mathcal{Z}, \lambda) : \mathbf{A}(view)[t_0, t_1] > \mathbf{C}(view)[t_0, t_1]\right] \leq e^{-\eta\beta t}$$

The conditions on parameters that guarantee protocol consistency are the following:

- There is a positive constant ϕ, such that for any $view \in \texttt{EXEC}^{\Pi_{sleepy}}(\mathcal{A}, \mathcal{Z}, \lambda)$ with non-zero support, for every $t \leq |view|$,

$$\frac{\text{alert}^t(view)}{N_{crupt}} \geq \frac{1+\phi}{1-2pN\Delta} \tag{1}$$

  where $\text{alert}^t(view)$ denotes the number of nodes that are alert at time t in *view*;

- there is some constant $0 < c < 1$ such that $2pN\Delta < 1 - c$.

$$2pN\Delta < 1 - c \tag{2}$$

These conditions are derived from the consistency proof shown in the Sleepy consensus paper [12], in particular they come from the proof for lemma 3. The above conditions can be translated in a more operational form into the following equation:

$$(1 - 2\alpha\Delta)\alpha > \beta \tag{3}$$

where $\alpha = p \cdot N_{alert}$ and $\beta = p \cdot N_{crupt}$.

4.3 Random variables method

Since the core of this paper, as we will see in the following chapters, is to improve the estimate made on convergence opportunities, let's try to understand the main ideas on how Pass et al., in [12] and previously in [11], proceeded to obtain their. The following results are a recast of the proof of lemma 2 of [12].

First, it is necessary to define a mathematical tool to determine the confidence intervals of the values assumed by random variables. The tool in question is the Chernoff bound.

Theorem 4 (Chernoff bounds). Let $\boldsymbol{X}_1, \dots, \boldsymbol{X}_n$ be a sequence of random variables such that $a \leq \boldsymbol{X}_i \leq b$ for all i. Let $\boldsymbol{X} = \sum_{i=1}^{n} \boldsymbol{X}_i$ and set $\mu = \mathbb{E}(\boldsymbol{X})$. Then, for all $\delta > 0$:

- Upper tail: $Pr[\boldsymbol{X} \geq (1+\delta)\mu] \leq e^{-\frac{2\delta^2\mu^2}{n(b-a)^2}}$;
- Lower tail: $Pr[\boldsymbol{X} \geq (1-\delta)\mu] \leq e^{-\frac{\delta^2\mu^2}{n(b-a)^2}}$.

Using the above theorem, it is possible to establish bounds with high probability; that is, the bound fails with probability that decreases exponentially in the number of rounds executed by the protocol.

To effectively count the number of convergence opportunities in Π_{ideal}, consider some view and imagine that $\mathcal{F}_{tree}$ flips a coin for each alert node in each round in order to determine the leader at round r. $\text{Alert}^r(view)$ flips will then be required. At this point, the following 3 random variables can be defined:

- Let $\boldsymbol{X}$ denote the total number of heads in all the alert coins during $[t_0, t_1]$. Due to the Chernoff bound, for any $\epsilon > 0$, it holds that

  $$Pr[\boldsymbol{X} < (1-\epsilon) \cdot \alpha t] \leq \exp\left(-\Omega(\alpha t)\right)$$

  Henceforth let $L := (1-\epsilon) \cdot \alpha t$ for a sufficiently small constant ϵ.

- Let $\boldsymbol{Y}_i = 1$ iff after the i-th heads in the alert coin sequence during $[t_0, t_1]$, there exists a head in the next $N_{alert}\Delta$ coin flips. Then $\boldsymbol{Y} = \sum_{i=1}^{L} \boldsymbol{Y}_i$. Basically $\boldsymbol{Y}$ counts how many times a head is followed by another head within Δ rounds. It's easy to see that its expected value is lees or equal than $pN_{alert}\Delta L = \alpha\Delta L$. By Chernoff bound, it holds that for any $\epsilon_0 > 0$,

$$Pr[\boldsymbol{Y} > \alpha\Delta L + \epsilon_0 L] \leq \exp\left(-\Omega(\alpha t)\right)$$

- Let $\boldsymbol{Z}_i = 1$ iff before the i-th heads in the alert coin sequence during $[t_0, t_1]$, there exists a head in the previous $N_{alert}\Delta$ coin flips. Then $\boldsymbol{Z} = \sum_{i=1}^{L} \boldsymbol{Z}_i$. Basically $\boldsymbol{Z}$ counts how many times a head is preceded by another head within Δ rounds. For the same reasons as $\boldsymbol{Y}$, using Chernoff bound, for any $\epsilon_0 > 0$,

$$Pr[\boldsymbol{Z} > \alpha\Delta L + \epsilon_0 L] \leq \exp\left(-\Omega(\alpha t)\right)$$

Observe at this point that when $\boldsymbol{X}_i = 1$, $\boldsymbol{Y}_i = 0$ and $\boldsymbol{Z}_i = 0$, a convergence opportunity occurs. Thus convergence opportunities will at least be all heads obtained minus the cases where a win occurred within delta rounds before and the cases where a win occurred within delta rounds after. Translated into formulas:

$$\mathbf{C}(view)[t_0 : t_1] \geq \boldsymbol{X}(view) - \boldsymbol{Y}(view) - \boldsymbol{Z}(view)$$

Now it is enough to combine the threshold values obtained through the Chernoff bound, considering *view* where bad events, in which the thresholds are not met, do not occur and obtain the final estimate on convergence opportunities, which is exactly what we stated in lemma 1. For each $\eta > 0$,

$$\begin{aligned} \boldsymbol{X} - \boldsymbol{Y} - \boldsymbol{Z} &\geq (1 - 2\alpha\Delta - 2\epsilon_0)L \\ &\geq (1 - \eta)(1 - 2\alpha\Delta)\alpha t \\ &\geq (1 - \eta)(1 - 2pN\Delta)\alpha t \end{aligned}$$

4.4 Markov chains method

We are going to apply a simple blockchain analysis framework proposed by Kiffer et al. in [6] that combines the Pass et al. approach mentioned above and Markov chains that allow us to capture the dynamics of the protocol with the presence of an adversary. Basically, each state of the Markov chain represents the state of the blockchain after an event of interest. These events will coincide from the outcome of the leader election on all nodes and thus in a possible addition of blocks to the chain or more or less prolonged periods of silence. These events will be represented by the edges connecting the various states of the Markov chain. The definition of the chain obviously depends on the consensus protocol studied and can be adapted to different types of protocols.

The Markov chains used in this type of analysis all have the following properties:

- *time-homogeneous*: the probability of transitioning from one state to another is only dependent on the states, and not on the time at which the transition occurs;
- *irreducible*: it is possible to get to any state from any other state;
- *ergodic*: every state is aperiodic and has a positive mean recurrence time.

This is because any chain that respects the 3 properties listed above has a (unique) stationary distribution π. Therefore, it makes sense to define, for each state v, the limit for n tending to infinity of the probability that the chain is in the state v after n state transitions, which we shall call $\pi(v)$. This allows us to define, for each sequence of T rounds, the number of times we expect to find the blockchain in the state v, which is $\pi(v) \cdot T$, for T sufficiently large. However, the exact number of occurrences of a state of interest could certainly deviate from the expectation. Fortunately, we can invoke the theorem 4 of the Chernoff bounds stated earlier, which allows us to obtain the concentration bounds of the event of interest, as long as we define the associated random variables appropriately. For instance, to measure the number of visits to a particular state v, we set $f_i(v)$ to 1 and set $f_i(u)$ to 0 for $u \neq v$, for all i. To measure the number of traversals of a particular edge, we add an auxiliary vertex in the middle of the edge, and set the measure to be the number of visits to the auxiliary vertex.

With the above necessary assumptions in mind, the method presented by Kiffer et al. consists of constructing the Markov chain associated with the consensus protocol first in a generic version containing only the main states that allow for the identification of convergence opportunities, then making the same chain more detailed by adding all possible states for each elective round. At this point, it is necessary to calculate some quantities in sequence. First, we must determine the probability of traversing each edge and the stationary distribution of the chain. These quantities will allow us to define the expected time spent on each edge from which we can subsequently derive the total weighted time spent on all edges. If the chain has been constructed properly, it will be possible to identify an edge that corresponds exactly to a convergence opportunity, so we can calculate how many convergence opportunities we expect by counting how many times that same edge will be crossed. The convergence opportunities will then be equal to the probability of crossing that edge from the correct state over the total weighted time spent on all edges, obviously all with confidence intervals derived from the Chernoff bound. In the chapter 5, you can see its application on the Sleepy consensus protocol.

5 Markov method application

5.1 Counting convergence opportunities

The consistency condition mentioned in chapter 4.1 is dependent on how convergence opportunities are calculated. This is because different calculation methods may lead to more or less precise estimates of their true value and thus influence the security parameters imposed on the protocol in order for it to be consistent.

5.1.1 The need for a new estimate

The calculation method presented by Pass and Seeman in [11] and reused in the Sleepy paper [12] in their count, leads to a certainly correct estimate of convergence opportunities but behaves as an underestimate in certain situations. Let's take the following sequence of rounds as an example:

$$W\ d\ W\ D\ W\ D\ W\ d\ W\ d\ W\ d\ldots$$

Where W means a successful alert leader election, d means a sequence of less than Δ rounds and D means a sequence of more or equal than Δ rounds. Note how there is a convergence opportunity given by $D\ W\ D$. The count must therefore be in the perfect case 1, but we still expect a value that is at least 0. On the other hand, applying the formula proposed in [12] and that we reviewed in section 4.3, we obtain that:

$$\mathbf{C}(view) \geq \mathbf{X}(view) - \mathbf{Y}(view) - \mathbf{Z}(view) = 6 - 4 - 4 = -2$$

Where $\mathbf{X}$, $\mathbf{Y}$ and $\mathbf{Z}$ are random variables defined as follows:

- $\mathbf{X}$ denotes the total number of successful alert leader elections in the sequence;
- $\mathbf{Y}$ denotes the number of times a victory is followed by another victory within Δ rounds;
- $\mathbf{Z}$ denotes the number of times a victory is preceded by another victory within Δ rounds.

This is a quick definition of the random variables used, for a more precise definition please refer to lemma 2 of [12] or our transposition in section 4.3. The outcome suggests that there may be room for improvement in the count.

5.1.2 New counting method

Let's try to apply the method proposed by Kiffer et al. in [6] and explained in the section 4.4. Start by defining the following two states:

S_0 := messy state where leaders are elected in less than Δ rounds,

S_1 := ordered state where there are quiet periods of more than Δ rounds between elections.

These states are the only ones relevant to the analysis we want to do because they allow us to identify convergence opportunities without making the Markov chain too complex to study. Therefore, bearing in mind the definition of convergence opportunities in the protocol under consideration and the two states just defined, we can construct the following Markov chain that captures all possible events after a sequence of multiple rounds of the Sleepy consensus protocol.

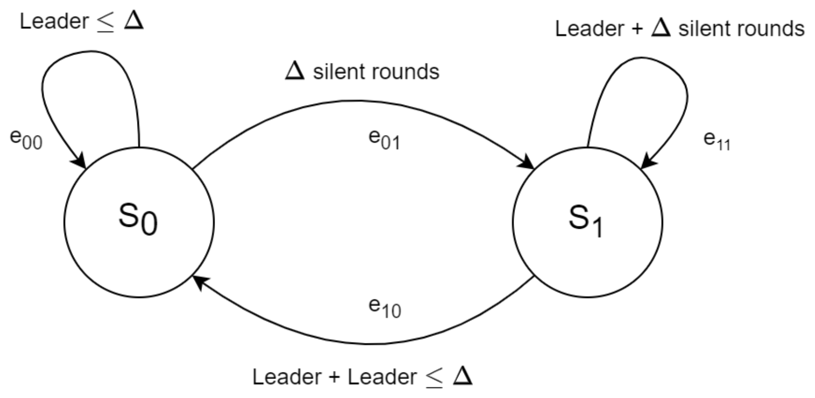

Figure 1: Sleepy protocol execution Markov chain

As the figure shows, we remain in S_0 if there are two elections within Δ rounds; as soon as we have a quiet period of at least Δ rounds we move to S_1; we stay in S_1 as long as leader elections are separated by quiet periods of at least Δ rounds; and finally we return to S_0 as soon as there is a new election before Δ rounds from the previous one. We call the transitions following the notation in the figure with e_{00}, e_{01}, e_{10} and e_{11} where e_{ij} denotes the event that allows the passage from the state S_i to the state S_j. We can expand the chain in figure 1 by representing each transition event e_{ij} in more detail. It is in fact sufficient to include the outcome of each individual instance of the leader election executed round by round and then construct the associated Markov chain containing a visualisation for individual rounds. We obtain the Markov chain presented in figure 2. We represented each specific path e_{ij} with a color in order to identify and distinguish them easily. The boxes contain instances of the leader election of $\Delta - 1$ rounds to describe the transition from one state to another within blocks of Δ rounds. This will help us identify the pattern of convergence opportunities and quantify the probabilities associated with transition paths more effectively.

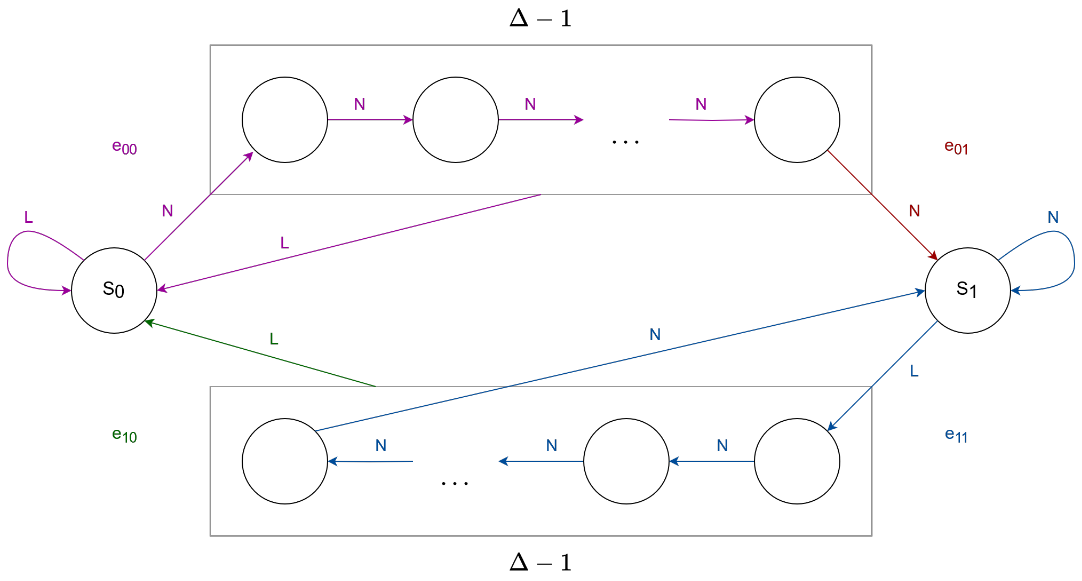

Figure 2: Sleepy protocol execution Markov chain round by round

N = round with no leader elected $\quad L$ = round with a successful leader election

We can now proceed to calculate some quantities that will be needed to reach the desired estimate for convergence opportunities. To begin with, we define the following variable:

$$h := p \cdot \mu$$

This will be a key parameter for the analysis. Since it is defined by the product of the probability of winning the leader election and the fraction of honest nodes in the system, h represents the probability that an honest node wins the leader election in a given round. The following equations will all be a function of this variable h, allowing us to express different concepts as dependant on the number of honest leaders. In addition, once the final estimate is obtained, it is easy to convert the variable h back to the fraction of corrupted and sleepy nodes in the system, through a straightforward change of variable. Now we are able to define the following quantity:

$P_\Delta :=$ probability of having Δ rounds without honest leader elections $= (1-h)^\Delta$

This is a fundamental probability since the occurrence of consecutive leaderless rounds (also called silent rounds) is a crucial event to study the model's Markov chain and to derive its transition probabilities. In fact we are able to define $Pr[e_{ij}]$ for each $i, j \in \{0, 1\}$ as a function of just P_Δ, which is again only a function of h.

$Pr[e_{00}] :=$ probability that e_{00} occurs, i.e. that there is at least one leader election within Δ rounds from the previous one $= 1 - P_\Delta$

$Pr[e_{10}]$:= probability that e_{10} happens, i.e. that there is at least one leader election within Δ rounds from the previous one $= 1 - P_\Delta$

$Pr[e_{11}]$:= probability that e_{11} happens, i.e. that the leader election is followed by Δ silent rounds $= P_\Delta$

$Pr[e_{01}]$:= probability that e_{01} happens, i.e. that the leader election is followed by Δ silent rounds $= P_\Delta$

At this point, it makes sense to define the probabilities of being in states S_0 and S_1, which we will respectively call π_0 and π_1, as the breakdown of the event into its components already known to us and then solve the simple linear system we obtain.

$$\pi_0 = Pr[S_0] = Pr[e_{00}] \cdot Pr[S_0] + Pr[e_{10}] \cdot Pr[S_1] = (1 - P_\Delta)\pi_0 + (1 - P_\Delta)\pi_1$$
$$\pi_1 = Pr[S_1] = Pr[e_{11}] \cdot Pr[S_1] + Pr[e_{01}] \cdot Pr[S_0] = P_\Delta \pi_0 + P_\Delta \pi_0$$

Since $\pi_0 + \pi_1 = 1$ being the certain event, we can easily deduce that

$$\pi_0 = 1 - P_\Delta$$
$$\pi_1 = P_\Delta$$

We now notice that the edge e_{11}, looping on state S_1, exactly captures convergence opportunities. When we transition from S_0 to S_1 we get a quiet period of Δ rounds, and whenever we loop in S_1 we get a successful leader election and another quiet period of Δ rounds. Thus looping in S_1 gives us a sequence of convergence opportunities. Therefore, to exactly count the expected number of honest convergence opportunities, we only need to count the expected number of times the edge e_{11} is traversed.
To do so, start by defining the two following notation:

ℓ_{ij} := expected time spent on the edge e_{ij}

$p_{i|\leq\Delta}$:= Pr[leader elected in round i conditioned by a quiet period of Δ rounds]

Since the probability of an event can be defined as favourable cases over all possible cases, it is easy to conclude that

$$p_{i|\leq\Delta} := Pr[\text{leader at time } i \mid \text{quiet period} \leq \Delta] = \frac{(1-h)^{i-1} \cdot h}{\sum_{j=1}^{\Delta} (1-h)^{j-1} \cdot h}$$

Furthermore, by looking at the associated Markov chain of figure 2, we can easily obtain the expected rounds spent on the edges:

$$\ell_{00} = \sum_{i=1}^{\Delta} i \cdot p_{i|\leq\Delta} \qquad \ell_{01} = \Delta$$
$$\ell_{11} = \frac{1}{h} + \Delta \qquad \ell_{10} = \frac{1}{h} + \sum_{i=1}^{\Delta} i \cdot p_{i|\leq\Delta}$$

From all the above equations, it follows that the total weighted time spent on all edges is given by the sum of the probability of being on each edge times the expected time spent on that edge times the probability of being in that state.

$$\mathcal{T} = \sum_{i,j} Pr[e_{ij}]\pi_i \ell_{ij}$$

The final result which allows to quantify the number of convergence opportunities is based on the law of large numbers and the stability of ergodic Markov chains. The law of large numbers implies that the relative frequency with which the chain visits a state S_i converges to the stationary probability π_i. In practice, if we observe the chain over a long period T, the number of times the chain is in S_i is approximately $T \cdot \pi_i \cdot Pr[e_{ij}]$. Dividing the total number of passes over e_{ij} by the observation period T, we obtain the transition rate: $\pi_i \cdot Pr[e_{ij}]$. At this point it makes sense to divide the expected number of passages on the path e_{ij} by the total weighted time of the chain $\mathcal{T}$ in order to normalise the number of passages on the path with respect to the total time required to travel all the paths of the chain. Considering the above reasoning for the path e_{11} it finally follows that the number of convergence opportunities is

$$\mathbf{C} = \frac{Pr[e_{11}]\pi_1}{\mathcal{T}} = \frac{P_\Delta^2}{\sum_{i,j} Pr[e_{ij}]\pi_i \ell_{ij}} \tag{4}$$

5.1.3 Concentration bounds

After obtaining the estimate, we can calculate its confidence interval by applying Chernoff's theorem on the bounds of random variables that we have stated in its generalised version in thm 4. To apply it to our new convergence opportunities estimate, however, it is useful to state it in its detailed version following the reference article [3]:

Theorem 5 (Chernoff-Hoeffding Bounds for Discrete Time Markov Chains)**.** Let M be an ergodic Markov chain with state space $[n]$ an stationary distribution π. Let $T = T(\epsilon)$ be its ϵ-mixing time for $\epsilon \leq 1/8$. Let $(V_1, \ldots, V_t)$ denote a t-step random walk on M starting from an initial distribution φ on $[n]$, i.e., $V_1 \leftarrow \varphi$. For every $i \in [t]$, let $f_i : [n] \to [0,1]$ be a weight function at step i such that the expected weight $E_{v \leftarrow \pi}\left[f_i(v)\right] = \mu$ for all i. Define the total weight of the walk by $X = \sum_{i=1}^{t} f_i(V_i)$. There exists some constant c (independent from μ, δ and ϵ) such that

1. $Pr\left[X \geq (1+\delta)\mu t\right] \leq \begin{cases} c\|\varphi\|_\pi \exp\left(-\delta^2 \mu t/(72T)\right) & \text{for } 0 \leq \delta \leq 1 \\ c\|\varphi\|_\pi \exp\left(-\delta \mu t/(72T)\right) & \text{for } \delta > 1 \end{cases}$

2. $Pr\left[X \leq (1-\delta)\mu t\right] \leq c\|\varphi\|_\pi \exp\left(-\delta^2 \mu t/(72T)\right) \quad \text{for } 0 \leq \delta \leq 1$

We can apply the theorem directly to the Markov chain in the figure 2 as it is an ergodic Markov chain with a stationary distribution $\pi = (\pi_1, \pi_2)$. Since we would like the random variable X to count the number of times the edge e_{11} is traversed, we can define the weight function $f_i(v)$ equal to 1 when v is a vertex belonging to path e_{11}, while we set it equal to 0 in case the vertex belongs to any other path. We then obtain the X describing the desired phenomenon and we can state that the number of passages over path e_{11} in T rounds is within the interval $(1 \pm \delta)$ of its expected value with probability $1 - e^{-\Omega(T)}$.

5.2 Results analysis

We would like to study how this new estimate behaves compared to the estimate in the Sleepy consensus paper [12] which follows the classical counting method introduced by Pass and Seeman in [11]. Specifically, what we are going to do is to study the performance of the difference between convergence opportunities and the number of adversarial time slots the adversary needs to break convergence opportunities and prevent the honest chain from converging to consensus. First, however, it is necessary that both consistency conditions that we have at our disposal are comparable, i.e. they are functions with the same variables. Therefore, we will now illustrate a series of results that allow for a simplification of the convergence opportunity formula 4, eventually resulting in an equation that is a function of the fractions of sleepy and corrupted nodes.

5.2.1 Markov estimate simplification

Let us first calculate some quantities that we will need in the simplification process. A factor that appears several times in $\mathbf{C}$ is the summation of the parameter $p_{i|\leq\Delta}$.

Lemma 6 ($p_{i|\leq\Delta}$ simplification)**.**

$$p_{i|\leq\Delta} = \frac{1-(1-h)^{\Delta}(1+\Delta h)}{h(1-(1-h)^{\Delta})}$$

Proof. From the definitions made in the chapter 5.1, we know that:

$$p_{i|\leq\Delta} = \sum_{i=1}^{\Delta} i \cdot \frac{(1-h)^{i-1}\cdot h}{\sum_{j=1}^{\Delta}(1-h)^{j-1}\cdot h} \tag{5}$$

Let's try to expand the numerator and denominator separately and obtain an equation without summation.

To deal with the numerator, one can use the known series $\sum_i ir^{i-1}$, so that:

$$h\sum_{i=1}^{\Delta} i\cdot(1-h)^{i-1} = h\left(\frac{1-(1-h)^{\Delta}}{h^2} - \frac{\Delta(1-h)^{\Delta}}{h}\right) = \frac{1-(1-h)^{\Delta}}{h} - \Delta(1-h)^{\Delta}$$

For the denominator similarly, you can use the geometric series of radius $1-h$:

$$\sum_{j=1}^{\Delta}(1-h)^{j-1}\cdot h = h\sum_{j=1}^{\Delta}(1-h)^{j-1} = h\cdot\frac{1-(1-h)^{\Delta}}{1-(1-h)} = 1-(1-h)^{\Delta}$$

Put it all together and you get that the initial summation can be simplified as follows:

$$(6) = \frac{\frac{1-(1-h)^{\Delta}}{h} - \Delta(1-h)^{\Delta}}{1-(1-h)^{\Delta}} = \frac{1-(1-h)^{\Delta}(1+\Delta h)}{h(1-(1-h)^{\Delta})}$$

□

Theorem 7 (Convergence opportunities simplification).

$$\mathbf{C} \approx \frac{1}{\Delta c}(1-\sigma-\rho)e^{-\frac{2}{c}(1-\sigma-\rho)}$$

Proof. Using lemma 6, we can redefine the expected rounds spent on the edges as follows, by making a straightforward substitution in their definition:

$$\ell_{00} = \frac{1-(1-h)^{\Delta}(1+\Delta h)}{h(1-(1-h)^{\Delta})} \qquad \ell_{01} = \Delta$$

$$\ell_{11} = \frac{1}{h}+\Delta \qquad \ell_{10} = \frac{2-(1-h)^{\Delta}(2+\Delta h)}{h(1-(1-h)^{\Delta})}$$

Replace everything within the equation for $\mathbf{C}$ we found in equation (4) and perform the necessary sums and simplifications. Since the equation will be very long, it may be useful to define the following quantities:

$$A = \frac{1}{h}\left[1-(1-h)^{\Delta}][1-(1-h)^{\Delta}(1+\Delta h)\right] \qquad B = \Delta(1-h)^{\Delta}[1-(1-h)^{\Delta}]$$

$$C = \frac{1}{h}\left[2(1-h)^{\Delta}-(1-h)^{2\Delta}(2+\Delta h)\right] \qquad D = (1-h)^{2\Delta}\left(\frac{1}{h}+\Delta\right)$$

By summing the above quantities all together, you can easily verify that $A+B+C+D = 1/h$. Therefore we obtain that:

$$\mathbf{C} = \frac{P_{\Delta}^{2}}{\sum_{i,j} Pr[e_{ij}]\pi_i\ell_{ij}} = \frac{(1-h)^{2\Delta}}{A+B+C+D} = h(1-h)^{2\Delta}$$

To conclude, note that $(1-x)^{\alpha} \approx e^{-\alpha x}$ for $x \in (0,1)$ and $\alpha \gg x$ as in this case. This will allow us to remove the exponential dependence on Δ (which makes the function almost impossible to analyse using classical numerical methods due to its very high order of magnitude being equal to 10^{13}).

$$\mathbf{C} \approx he^{-2\Delta h}$$

At this point, since we want the dependence on σ and ρ, it will be enough to perform the following variable substitution according to the definitions of h and p:

$$h = p\mu = \frac{1}{\Delta c}(1-\sigma-\rho)$$

Therefore

$$\mathbf{C} \approx \frac{1}{\Delta c}(1-\sigma-\rho)e^{-\frac{2}{c}(1-\sigma-\rho)} \tag{6}$$

□

This allows us to study the new consistency condition as the adversary's elective power varies for a fixed fraction of sleeping nodes.
The final consistency condition using this new estimate will be defined as follows:

Theorem 8 (New final consistency condition)**.** The Sleepy protocol new consistency condition using the latest convergence opportunities counting method is the following:

$$(1-\sigma-\rho)e^{-\frac{2}{c}(1-\sigma-\rho)} - \rho > 0 \tag{7}$$

Proof. To prove consistency, we require that convergence opportunities are strictly greater than the adversarial time slots that are required to break all convergence opportunities in a time interval, so we will have to study the following condition

$$\frac{P_\Delta^2}{\sum_{i,j} Pr[e_{ij}]\pi_i \ell_{ij}} > \beta \tag{8}$$

At this point, it is enough to replace the left-hand side of the inequality with the simplification we found in equation (6) and rewrite β with the definition we gave in the chapter 3 to obtain the desired equation. □

5.2.2 Pass and Seeman estimate change of parameters

Before comparing it with the old condition 3 determined by Pass and Seeman in [11], we should rewrite 3 according to the same variables used in our new estimate. To do so, we use the defintions of α and β but pay attention that in the Sleepy withepaper [12], following the approach of [11], p is defined as $\frac{1}{\Delta N c}$ and then they use pN as probability of mining a node. So basically the two models are the same but we have to simplify the parameters with this alterated version of p:

$$\beta = p\rho N = \frac{\rho}{c\Delta}$$

$$\alpha = p\mu N = \frac{(1-\sigma-\rho)}{c\Delta}$$

Now we can substitute the values of α and β in $(1-2\alpha\Delta)\alpha > \beta$ and obtain the final condition of consistency with the old estimate of convergence opportunities:

$$\left(1 - \frac{2(1-\sigma-\rho)}{c}\right) \cdot (1-\sigma-\rho) - \rho > 0 \tag{9}$$

5.2.3 Conditions comparison

We solve the two equations (7) and (9) by finding the maximum ρ for which the consistency condition is still satisfied as c varies and for fixed values of σ. This allows us to work out which value of p can guarantee consistency against the adversarial elective power fraction ρ. The solutions were found using the Brent's search method which is a hybrid linear search method that utilises the bisection method, the secant method and quadratic interpolation. The python script that was used to perform the numerical analysis can be found in Appendix A. The following graph summarises the results by showing the trend of the maximum ρ guaranteeing consistency as the parameter c varies for both the compared conditions. The slider allows the fraction of sleepy nodes in the network to be changed, which leads to further variation in the solutions of the conditions.

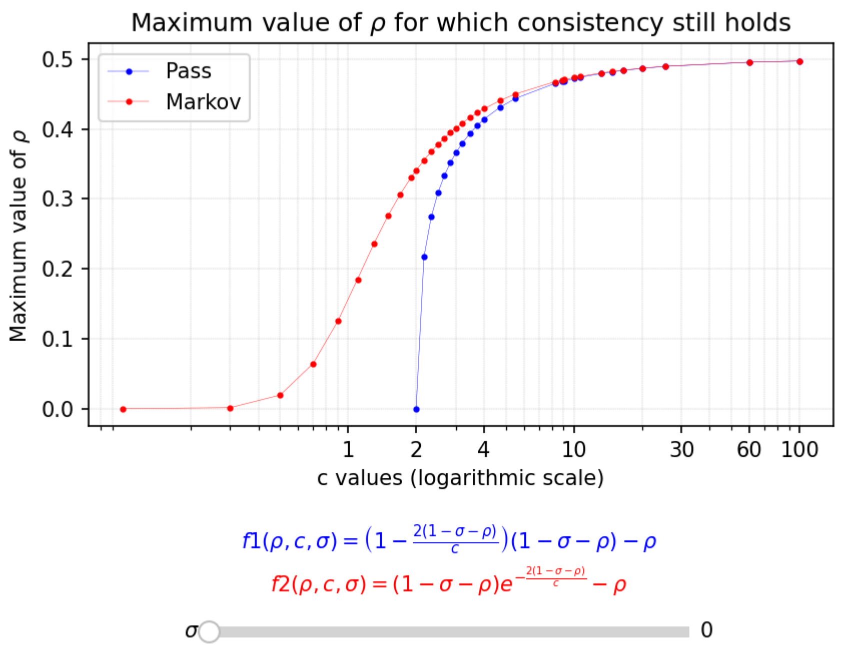

Figure 3: Comparison of conditions with $\sigma = 0$

In this first image 3, we reported the output of the script, which is the graph of the solutions of the two conditions, when the fraction of Sleepy node is zero and for a set of c values with a particular distribution that allowed us to compare both the functions in a clear way. As you can see, the new estimation allows for a consistency condition that provides greater resistance against a higher adversarial elective power at the same value of c and thus an equal probability of victory for the leader election. In addition, it allows consistency to be defined for values of c less than 2. More generally, the new estimate allows us to define a condition that performs better for small values of c, the difference between the two conditions then decreases as c increases.

Let's see what happens when we change the value of σ by adding a small fraction of sleepy nodes.

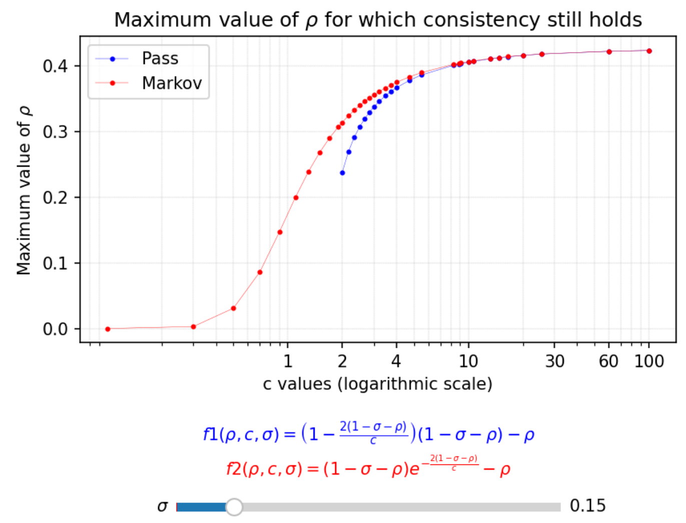

Figure 4: Comparison of conditions with $\sigma = 0.15$

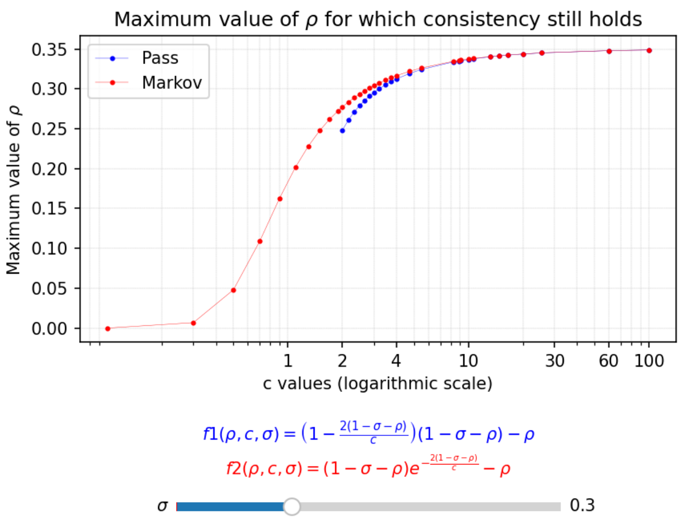

Figure 5: Comparison of conditions with $\sigma = 0.3$

It happens what we expected, that is a reduction in the threshold of tolerance of the adversarial elective power. From the figures 4 and 5, in the cases where σ is non-zero, we can identify the decreasing trend of the tolerable ρ_{max} as σ increases. This is logical since, as the fraction of sleepy nodes increases, the fraction of corrupted nodes acquires more power against the network, causing the system to be less tolerant towards the adversary. We can also see that as σ increases, the previous condition tends towards the new consistency condition. Given the dependence for both conditions on σ and c, the previous graphs have been developed by using a slider on the variable σ, fixing its value and studying the behaviour of the ρ solutions as the parameter c varies. However, it is possible to represent the solutions three dimensionally in order to observe their shape as both σ and c vary. The result will be the pair of surfaces shown in the figure:

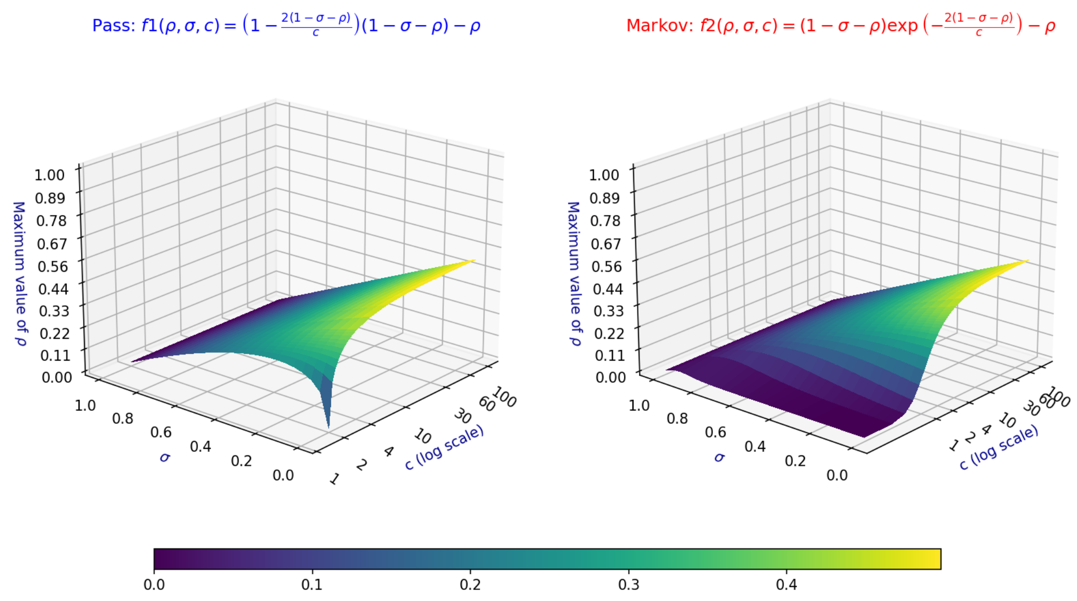

Figure 6: Conditions solutions on a 3D graph

The solutions represented as three-dimensional surfaces correctly maintain the same behaviour as their two-dimensional counterparts. The condition obtained from the new estimate of convergence opportunities is defined across the whole portion delineated by $0 < c < 2$ and $\sigma \in (0, 1)$ unlike the standard condition. In addition, the surface area of the new condition entirely covers the surface area of the previous condition, which means that by configuring the Sleepy consensus protocol with those parameters, the system is able to tolerate a larger adversarial elective power. To obtain the graph in figure 6, I used a variation of the Python script explained in Appendix A by not limiting the values of σ. I will not comment on the details as I did for the 2D script but the code is still freely available in the GitHub repository linked in Appendix A. The following table compares, for specific values of c and σ, the fraction of corrupted nodes supportable for both consistency conditions and shows the improvement obtained with the new convergence opportunities estimate. It is evident that the greatest improvement is obtained for low values of c but the result could also be significant for larger values of c since, for networks with a large number of nodes (e.g., $N = 10^5$), the new condition would allow thousands of more corrupted nodes to be supportable.

Table 3: Percentage improvement for specific c values ($\sigma = 0.1$ and $\sigma = 0$)

σ	c	ρ_{max} for f_1	ρ_{max} for f_2	Improvement (%)
0.1	3	0.3487	0.3715	6.56
0.1	10	0.4277	0.4288	0.26
0.1	60	0.4466	0.4466	0.01
0	3	0.3660	0.4016	9.71
0	10	0.4721	0.4737	0.33
0	60	0.4958	0.4958	0.01

6 Sleepy best attack analysis

At this point, knowing the values of the security parameters and the number of corrupted nodes tolerable to satisfy consistency, it would be useful to study how the adversary can counter the consistency property anyway, delaying the reaching of consensus as much as possible. In this chapter we will then address the best attack the adversary can adopt against the network with the aim of potentially creating an alternate chain and sustain it in paraller with the actual one. Doing so, the honest nodes of the network would have different chains as their local views and the consistency property would not be satisfied. The following analysis is useful to evaluate the probability of this scenario to happen and how many blocks should we wait before considering the chain to be finalized.

6.1 Adversary capabilities

First, lets remark the capabilities of the adversary in this protocol to understand how it can efficiently exploit all the controlled corrupt nodes. Basically, the Sleepy protocol is nothing more than the Proof of Work protocol in which the solving of computationally difficult crypto puzzles is replaced by a leader election of a random oracle. Each node has the same probability of being elected leader at a given time instant t, however, by eliminating from the block creation phase the intensive work that serves as the security system of the protocol itself, the adversary is given the possibility to mine a block and reuse it on another chain with no effort. This is given by the fact that the process of obtaining the right to publish the block is independent from the creation of the block itself, and thus a corrupt node that has obtained the leader role in a round t, can use its malicious block on several chains simultaneously in round t, without having to mine it on each chain.

In addition, as can be seen from the algorithm of $\mathcal{F}_{\text{tree}}(p)$, corrupt leaders have a special call to the `extend`$(chain, B, t')$ function:

Algorithm 4 `extend`$(chain, B, t')$

```
On receive extend($chain, B, t'$) from corrupt party $\mathcal{P}^*$: let $t$ be the current time
    assert $chain \in tree$, $chain \| B \notin tree$, leader($\mathcal{P}^*, t'$) output 1
        assert time($chain$) $< t' \leq t$
            append $B$ to $chain$ in $tree$, record time($chain \| B$) $:= t'$, and return $succ$
```

The honest node that receives the malicious block with timestamp t' from a corrupt leader, first checks that it has not received it already and then checks if in round t'

the node creating the block was indeed the leader. Finally it adds the new block to its local view chain. This allows the adversary to potentially add blocks in past rounds, as long as the timestamp of the blocks is strictly increasing.

In addition the presence of sleepy nodes decreases the percentage of honest active nodes in the network and further slows down the achievement of consensus. It is therefore evident how the Sleepy protocol adversary is potentially stronger than the PoW protocol adversary. This is why a completely different demonstration was needed to show the consistency property of the protocol and special care will be needed in modeling the best attack against the network.

6.2 Attack model

Let's begin by recalling that the adversary is responsible for message deliveries in the network: potentially, broadcasts from honest nodes of their local views could be delayed up to a maximum of Δ rounds. However, once the message has been sent and the maximum delay has passed, it must necessarily be delivered to the recipient node. Consequently, if the adversary wants to maintain an inconsistent view of the chain for various honest nodes, it has to maintain false the definition 4.1. In particular, there exist at least a couple of honest nodes, i and j, for which their common prefixes are different, i.e., given their two local views $chain_i^t(view)$ and $chain_j^{t'}(view)$ for some time rounds $t \leq t'$, $\texttt{common\_prefix}_i^t(view)$ and $\texttt{common\_prefix}_j^{t'}(view)$, which are basically the first $\ell = |chain_i^t(view)| - T$ blocks of their local views, differ by at least one block.

Potentially the attacker could maintain the nodes as isolated as possible by making their local views all different within the Δ rounds of the network delay. However, this would make the attack unnecessarily complex since only two nodes with a different common prefix are needed to break the consistency property. Our goal is to construct the best attack to the system by giving all possible advantages to the adversary so that, once the probability of successfully executing it is determined, the result is the worst case scenario for the Sleepy protocol and we could configure the security parameters appropriately. As a result, we can divide the set of honest nodes into two subsets of equal size, and the adversary's goal will be to keep the two local views of the two groups different to slow down the achievement of consensus. This situation occurs naturally, without the need for an external malicious intervention, upon the occurrence of a fork. A fork is a configuration of the main chain for which the entire network has the same local chain until the round $t-1$, while at round t, a fraction of the nodes considers one block as the valid one and the remainder of the nodes another. To make this happen, it is sufficient that within Δ rounds, two honest nodes are elected leaders and each of them creates and proposes a block. Due to any inefficiency of the communication system, the broadcasts immediately reache two disjoint sets of honest nodes and the consistency property is momentarily unsatisfied. The adversary will then only need to maintain the two branches of the forked chain for as many rounds as possible. This necessarily implies feeding the two chains created in parallel so that they maintain the same number of blocks. The attack just described is called *fork sustain attack* and we will try to model it using Markov chains, following the procedure of Kiffer et al. on the GHOST protocol presented in the reference paper [6].

6.2.1 Common patterns

Considering the outcome of the leader election on every round, the system could have an honest leader, a corrupt leader or a silent round. In case one of the above three events occurs, the system will assume different configurations and the adversary will have to behave in a certain way to continue the attack. We will refer to the two fork chains by the terms *right chain* (r) and *left chain* (l). Similarly, the leader elected from the subset of honest nodes with local view the right chain will take the name *right leader* and vice versa for the *left leader*. On the contrary, since corrupt nodes are controlled by the adversary, they are able to add blocks in both chains according to the convenience of the situation. We denote the probability of a right leader being elected with the notation h_r, the probability of a left leader being elected as h_l, and the probability of a corrupt leader being elected as a. Let's study the possible consequences assuming that the system starts from a parity situation where the two chains of the fork have the same length (i.e., the initial state of the fork creation). We can denote this state by the notation P.

1. *honest leader election*: in this case one of the two chains will be incremented by one block and an unbalanced state is reached. Depending on which of the two classes of leaders was elected, we can go to the state $r + 1$, in case of h_r, or $l + 1$, in case of h_l. From here within the next Δ rounds, the adversary must be able to counterbalance the fork by adding the missing block to the shorter chain.

2. *corrupt leader election*: in this case the adversary will not benefit from adding the block to a chain since this would lead to an imbalance of the fork. It will therefore keep the slot unused but reserve the option to use it in the future, counterbalancing the shorter chain in case of future honest elections. We therefore take up the concept of adversarial slots introduced in lemma 2, intended as past rounds in which the adversary won the leader election and through the malicious call of `extend`, the corrupted node can propose a block with a past timestamp.

3. *silent round*: having no election in a round keeps the configuration of the system unchanged. This event benefits the adversary in a situation of parity since the

fork status persists and consensus is delayed but in the presence of an imbalance it is convenient for the fraction of honest parties given that it would force the adversary to use a past slot and decrease its resources to fuel the fork.

Moreover, given a state of fork imbalance, the system can return to a state of parity, or at least move one step closer to it, only through an adversarial election, either an election of a shorter chain leader or through the spending of an adversarial slot by the adversary.

6.2.2 Tree graph

Following the possible patterns just described, it is easy to construct a tree graph showing the possible configurations of the forked chain and allowing us to visit its various states.

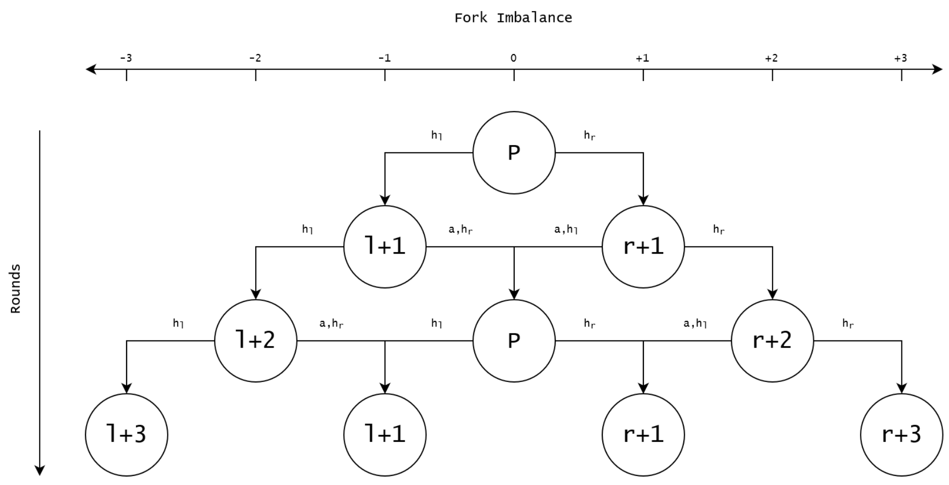

Figure 7: Tree graph of the attack states

As can be seen from the graph, we can conventionally associate the positive sign with the right chain and the negative sign with the left chain and potentially model the game through a random walk on the set of rational numbers. We start from 0, the state in which we are in P, and each round we have h_r probability of moving right and h_l probability of moving left. As soon as we are in a nonzero state, the probability of moving from i to $i - \frac{i}{|i|}$ (which is simply the rational number one step closer to zero starting from i) will be the union (logic `OR`) of a and h_r if $i \in \mathbb{Z}_{<0}$ or of a and h_l if $i \in \mathbb{Z}_{>0}$.

6.2.3 Model symmetry

The above sign convention is totally invertible given the symmetry of the fork structure and the attack that the adversary wants to sustain. Especially, with the purpose of building a mathematical model of the attack that is as simple as possible to solve, we are not interested in distinguishing between the two chains: what is relevant is in fact only the information whether we are in a parity state or in an unbalanced state in which one of the two chains has at least one block more. This is also a consequence of the adversary having equally divided the honest nodes into two equal subgroups and thus $h_r = h_l$. This choice is obviously the most advantageous for conducting the attack since with any other type of unequal subdivision, one of the two groups would certainly be more likely to be elected leader and the respective chain would receive new blocks at a faster rate than the other, with the risk that the adversary would not be able to counterbalance as quickly. Thus, dividing the set of honest parties into subsets of equal numbers allows the speed of growth of the chains to be limited as much as possible without advantaging one over the other.

6.3 Pseudocode

At this point we have all the elements to construct the pseudocode of the attack.

Algorithm 5 Fork sustain attack

```
1:  while fork persists do
2:      if honest leader elected then
3:          $\Delta$ countdown begins
4:          if honest leader elected on opposite chain OR $\mathcal{A}$ leader then
5:              fork is balanced
6:          else if $\Delta$ silent rounds then
7:              if $s \geq 1$ then
8:                  $s--$
9:                  fork is balanced
10:             else
11:                 A loses
12:             end if
13:         else if honest leader on the same chain then
14:             if $s \geq 1$ then
15:                 $s--$
16:                 $\Delta$ countdown restarts
17:             else
18:                 A loses
19:             end if
20:         end if
21:     else if $\mathcal{A}$ leader then
22:         $s++$
23:     end if
24: end while
```

When a fork occurs, the leader election instance is executed round by round and the relevant outcomes are *honest leader elected* or *adversary elected*. As soon as an honest node is elected leader, one of the two chains is incremented and a countdown of Δ rounds starts within which the adversary must counterbalance the fork. The fork is balanced only if, within the Δ rounds, an honest node of the opposing chain or a corrupt node is elected. If this does not happen, a series of leaderless rounds (called silent rounds) may follow or a further election may have occurred on the chain already in the lead: in both cases, the adversary will be forced to spend one of its past slots (indicated by the counter s). Note that using a past slot in the real protocol is not always possible since for an honest node to accept the block of the corrupted node, the timestamp of the block must be between the time of the chain and the current time round. In addition, blocks are linked via a hash chain to the previous block so it would be particularly difficult for the adversary to insert the block between two or more blocks in the chain. However, we simplify the model by allowing the adversary to use any past slot without restriction, this ensures that the attack studied is as critical as possible and the final results will be constructed considering the worst case scenario. If the variable s reaches 0 and there is a need to further decrease it, the opponent loses and the game ends. The variable s is incremented only in the case of a corrupt election in the parity state of the fork. Note how in line 16, the countdown is restarted after a slots decrease: this happens because in the case in which one of the two chains goes ahead by two blocks, we ask the opponent to immediately spend one slot to bring it back to the +1 state and it will need to counterbalance only the last inserted block.

6.4 Attack Markov chain

At this point it is easy to translate the pseudocode into the Markov chain that best describes the game. First, however, let us clarify some notation matters by summarizing the main parameters of the problem and introducing new ones that simplify the representation of the model. Given N the total number of nodes in the network, μ is the fraction of honest nodes, σ the fraction of sleepy nodes, and ρ the fraction of corrupted nodes. Recall that we are studying the Sleepy protocol in the case of static corruptions so the concentrations of node classes are fixed in time and known a priori. Δ is the maximum network delay, c is the average block time and p is the probability of winning the leader election in round t for any generic node. From the model formalization we know that $p = \frac{1}{\Delta c}$. Consequently, denote by a the probability that in a generic round, the adversary wins the leader election. Similarly, we call h the probability that in a generic round the honest parties win the leader election and denote by h_{-} the probability that in a generic round one of the two halves of the honest parties (right or left chain) wins the leader election. Given these basic probabilities, we denote by a^{Δ} the probability that there will be a corrupt election within Δ rounds, $h\_^{\Delta}$ the probability that there will be an election of a right or left leader within Δ rounds, and we will denote by $a^{\Delta}, h\_^{\Delta}$ the union of the two events. Finally $\Delta silent$ will indicate the probability of no leader being elected in Δ rounds. The states we will need in the Markov chain are the following:

$$P(s) = \text{state of fork parity in which } \mathcal{A} \text{ has } s \text{ adversarial slots}$$
$$\Delta(s) = \text{state of fork imbalance in which } \mathcal{A} \text{ has } s \text{ adversarial slots}$$

$$\begin{tikzcd}
\Delta(s-1) & & \\
& \Delta(s) \arrow[ul, bend right, "h\_^{\Delta}"'] \arrow[dl, bend right, "\Delta\ silent"'] \arrow[d, bend right, "a^{\Delta}, h\_^{\Delta}"'] & \\
P(s-1) \arrow[r, "a"] & P(s) \arrow[u, bend right, "h"'] \arrow[r, "a"] & P(s+1)
\end{tikzcd}$$

Figure 8: Best attack model Markov chain

It is important to point out, for the purpose of properly valuing the probabilities of the transitions between states, that the states of the Markov chain just proposed, aggregate the fork situation over several rounds. More precisely, the state $P(s)$ contains all the states in which the fork remained in a parity situation until there was a successful leader election. In this case, with probability μ we go into the state $\Delta(s)$ and with probability ρ we go into the state $P(s+1)$. This is because $P(s)$ contains a sequence of i execution of the consensus algorithm, in which $i-1$ rounds are leaderless and the i^{th} round has a successful leader election. Indeed, we are not interested in counting how many rounds the adversary remains in $P(s)$ since, as long as the fork has chains of equal length, consistency is not met and the game can proceed. The consequence of this definition of $P(s)$ is that the parameters a and h will depend only on the concentrations of honest and corrupt nodes and not also on p, since we do not care how frequent leader elections are, but just with what probability they are honest or corrupt. In the $\Delta(s)$ state, on the other hand, there is a countdown of exactly Δ rounds and the amount of round the adversary stays in this state is absolutely relevant. Hence, the probabilities of state transitions starting from $\Delta(s)$, will have the dependence on p. Therefore, the parameters assume the following values:

$$\Delta silent = (1-p)^{\Delta} \approx e^{-\frac{1}{c}}$$

$$a = \rho \qquad h = \mu \qquad h_{\text{-}} = p \cdot \frac{\mu}{2}$$

$$a^{\Delta} \approx 1 - e^{-\frac{\rho}{c}} \qquad h_{\text{-}}^{\Delta} \approx 1 - e^{-\frac{\mu}{2c}} \qquad a^{\Delta}, h_{\text{-}}^{\Delta} \approx 1 - e^{-\frac{1}{c}\left(\rho + \frac{\mu}{2}\right)}$$

The Markov chain presented in the figure 8 is a direct transposition of the pseudocode of the attack. Starting from the parity state $P(s)$, with probability h, it goes into the imbalance state $\Delta(s)$ and the countdown of Δ rounds starts. If $a^{\Delta}, h_{\text{-}}^{\Delta}$ happens within this time interval, it returns to $P(s)$ and the fork has increased its length by 1 (since both of its chains have received a new block). If, on the other hand, $h_{\text{-}}^{\Delta}$ happens, the adversary spends a slot, the fork increases its length by 1, and it remains in the unbalanced state but with one less slot. Finally, if $\Delta silent$ happens, the adversary is forced to spend a slot of its own and it returns to the state of parity with one less slot. If, in any parity state, there is an adversarial election, the system remains in the parity state but the adversarial slots are increased by 1. Note that, in visiting the various states of the Markov chain, the only paths that increase the length of the fork are $\Delta silent$, $h_{\text{-}}^{\Delta}$ and $a^{\Delta}, h_{\text{-}}^{\Delta}$. This is relevant since our goal is to count with what probability, starting from the parity state $P(s)$ and with s slots available, the adversary succeeds in feeding the fork for a length of at least k blocks. Let's call this quantity $P_s(k)$ and define some relation which allows us to define its values as a function of the parameters s, k and c. Similarly to $P_s(k)$, we can define $\Delta_s(k)$ which is basically the probability that, starting from the imbalanced state $\Delta(s)$ and with s slots available, the adversary succeeds in feeding the fork for a length of at least k blocks. To recap:

$$P_s(k) = Pr[\text{starting from } P(s), \mathcal{A} \text{ creates a fork of} \geq k \text{ blocks}]$$
$$\Delta_s(k) = Pr[\text{starting from } \Delta(s), \mathcal{A} \text{ creates a fork of} \geq k \text{ blocks}]$$

Considering the termination conditions of the game, we can define the following relations useful in solving the Markov chain:

$$P_s(k) = 1 \quad \text{if} \quad s \geq k$$
$$\Delta_s(k) = 1 \quad \text{if} \quad s - 1 \geq k$$
$$P_s(k) = 0 \quad \text{if} \quad s = -1$$

At this point it is easy to determine the solving equations of the chain:

$$P_s(k) = h \cdot \Delta_s(k) + a \cdot P_{s+1}(k) \tag{10}$$

$$\Delta_s(k) = (a^{\Delta}, h_{\text{-}}^{\Delta}) \cdot P_s(k-1) + h_{\text{-}}^{\Delta} \cdot \Delta_{s-1}(k-1) + \Delta silent \cdot P_{s-1}(k-1) \tag{11}$$

Replacing the probabilities with their respective values as a function of the parameters μ, ρ and c, we obtain:

$$P_s(k) = \mu \cdot \Delta_s(k) + \rho \cdot P_{s+1}(k) \tag{12}$$

$$\Delta_s(k) = \left(1 - e^{-\frac{1}{c}\left(\rho+\frac{\mu}{2}\right)}\right) \cdot P_s(k-1) + (1 - e^{-\frac{\mu}{2c}}) \cdot \Delta_{s-1}(k-1) + e^{-\frac{1}{c}} \cdot P_{s-1}(k-1) \tag{13}$$

6.5 Equation solution and results analysis

At this point, we can proceed with solving the equations (12) and (13). The system consists of two recursive equations easily solvable in a python script that you will find commented in annex B. The purpose is to determine, as the concentrations of nodes in the network ρ, σ and μ vary, the solutions of $P_0(k)$ for a set of c that may be of interest to us and for reasonable values of k. As an initial condition, we obviously set $s = 0$ since we want to obtain the probability of the attack succeeding as soon as the fork is created, without the adversary immediately having slots to use. We will then obtain the probability that the adversary has of performing a fork sustain attack by bringing the fork to length of at least k, starting with 0 adversarial slots. We can use the obtained sustain attack probability values to determine the T-parameter of the T-consistency definition, stated in the 4.1 definition.

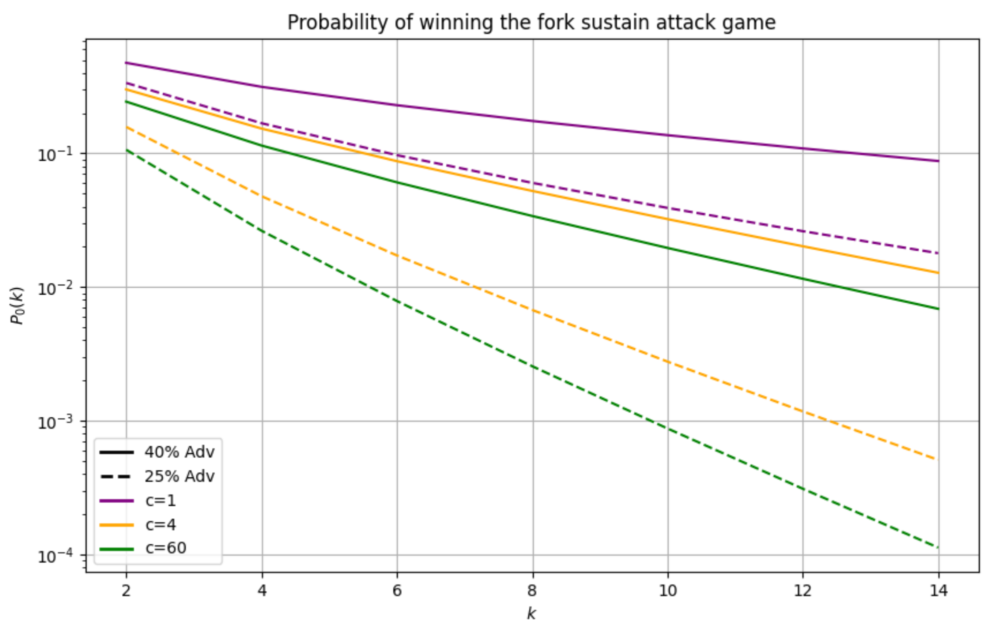

Figure 9: Fork sustain attack probabilities

In the graph in figure 9, you can see the output of the script, which shows the trend of $P_0(k)$ as k varies and for a fraction of adversary nodes ρ equal first to 0.25

(represented with a dashed line) and then to 0.4 (represented with a solid line). As we expected, for small values of c, there is a non-negligible probability of the attack succeeding even for values of k greater than 10. Thus taking a $c = 60$ favors the maximum fraction of corrupted nodes tolerable (as a consequence of the analysis done in chapter 5) and in addition would make the fork sustain attack more difficult to execute. Note that, again from the previous analysis of the consistency conditions performed in chapter 5, when $c = 1$, for the protocol to meet the consistency property, the fraction of corrupted nodes needs to be at most 0.15 (assuming total absence of sleeping nodes). Consequently, the cases considered with $c = 1$ describe the attack performed in a nonconsistent protocol: in practice, in this context the adversary would be able to sustain the fork for several blocks with a very high probability and could potentially prevent consensus reaches throughout the entire execution of the simulation. We now proceed to calculate the value of T-consistency by taking $\epsilon = 10^{-6}$ as the acceptable probability for the realization of the attack. This means that, when the consistency condition is met, given the values of the parameters ρ, σ, μ and c, we are sure with probability $1 - \epsilon$, that given the local chain of our view, all blocks minus the last T, can be considered finalized. This is because in Sleepy protocol executions, when consistency is respected, it is difficult for the adversary to sustain a fork for more than T blocks with probability larger than ϵ. By solving $P_0(k)$ as k varies until we obtain a value less than ϵ, we obtain the following results:

Table 4: T-consistency parameter values for $\sigma = 0.1$ and $c = 60$

ρ	μ	T
0.25	0.65	24
0.40	0.50	52

7 Conclusions

Thanks to this analysis, we obtained a new consistency condition that allows us to extend the domain of the previous one to values of c for which it was previously undefined and capable of supporting a higher fraction of corrupted nodes. In addition, we tested how the Sleepy consensus protocol behaves against its best attack, which allowed us to identify a model for determining the T-consistency parameter T.

The main critical issue in applying the framework proposed by Kiffer et al. in [6], was removing the Proof of Work concept and introducing the leader election, the core of the Sleepy protocol. Once this adaptation was made, the framework proved to have excellent potential and high flexibility in studying different aspects of the protocol. We are sure it can be a useful tool to analyze additional consensus protocols and even different blockchains properties.

7.1 Limitations and future work

Our analysis was conducted under specific simplifying assumptions. One among them is having applied the models of the proposed framework on an simpler version of the Sleepy protocol i.e. in a static corruption environment, as pointed out several times during the illustration of the models. The natural continuation of this analysis would then be to refit the framework in an environment with adaptive corruptions and adaptive sleepiness in which the various fractions of corrupted and sleeping nodes vary over time and are not known a priori. In addition, following the approach taken by the Sleepy's whitepaper [12], all theoretical analysis were performed in an ideal version of the consensus protocol in which the leader election is performed by a random oracle. In the actual implementation of the protocol, the leader election is carried out by the use of a public key infrastructure (PKI), a Common Random String (CRS) and is proven secure in the timing model of Dwork-Naor in [4]. In the whitepaper, Pass et al. then show how there is indistinguishability between the two models (the ideal one and the real one) through an hybrid protocol, however, it would still be useful to verify what happens, with the new consistency condition, in the real protocol and whether it might result in changes to the demonstration of indistinguishability or in the definition of security parameters.

We hope that the techniques and approaches developed in this thesis, particularly the use of Markov chains to prove the consistency property, can be applied to other consensus protocols beyond Sleepy. As the field of distributed systems and blockchain technology continues to evolve, research efforts will be crucial in improving protocol design, ensuring security under increasingly sophisticated adversarial models, and pushing the boundaries of what is possible in decentralized systems.

A Brent search script

Here you can consult the commented code of the script used to solve the equations on the consistency conditions. The code can be consulted in its totality and without comments in the following GitHub repository: Videars/Blockchain-Consistency[1].

```
import matplotlib.pyplot as plt
import numpy as np
from scipy.optimize import brentq
from matplotlib.widgets import Slider
```

Let's start by importing some essential libraries that will allow us to solve the problem we are dealing with. *Matplotlib* to represent graphs, *Numpy* to perform numerical operations and manage arrays, *Brentq* to determine the roots of functions using Brent's method and *Slider* to add interactive elements to the graph.

```
def f1(x, c, s):
    return (1 - (2 * (1 - s - x)) / c) * (1 - s - x) - x

def f2(x, c, s):
    return (1 - s - x) * np.exp(-(2 * (1 - s - x)) / c) - x
```

Let us define the two functions of interest. In the code, *f1* corresponds to 9 and *f2* corresponds to 7.

```
c_values_between_0_and_2 = np.linspace(0.1, 1.9, 10)
c_values_between_2_and_3 = np.linspace(2, 3, 7)
c_values_between_3_and_4 = np.linspace(3.2, 4, 4)
c_values_above_4 = np.random.exponential(scale=20, size=10) + 4
c_values_specific = np.array([10, 20, 60, 100])
c_values_1 = np.unique(np.concatenate([c_values_between_2_and_3,
c_values_between_3_and_4, c_values_above_4, c_values_specific]))
c_values_2 = np.unique(np.concatenate([c_values_between_0_and_2,
c_values_1]))
c_values_1.sort()
c_values_2.sort()
```

Then we build an array for the values of c, making sure that its distribution is chosen to be denser near lower values and sparser as it increases. In particular more points close to 2 for fine granularity and slightly fewer points as we move away from 2 using an exponential distribution. We made sure specific important points are included, then simply concatenate and sort unique values of c for both the functions.

```
lower_bound = 0
upper_bound = 1
```

We defined the search limits for the root finding algorithm with 0 and 1 since we know a priori the solutions must be included in this range due to the nature of ρ.

[1]https://github.com/Videars/Blockchain-Consistency

```
def update(val):
    s = slider_s.val
    x_max_values_f1 = []
    x_max_values_f2 = []

    for c in c_values_1:
        try:
            max_x_f1 = brentq(f1, lower_bound, upper_bound,
            args=(c, s))
            x_max_values_f1.append(max_x_f1)
        except ValueError:
            x_max_values_f1.append(lower_bound)

    for c in c_values_2:
        try:
            max_x_f2 = brentq(f2, lower_bound, upper_bound,
            args=(c, s))
            x_max_values_f2.append(max_x_f2)
        except ValueError:
            x_max_values_f2.append(lower_bound)

    ax.clear()
    ax.plot(c_values_1, x_max_values_f1, marker='o', markersize=2,
    linestyle='-', linewidth=0.2, color='b', label='Pass')
    ax.plot(c_values_2, x_max_values_f2, marker='o', markersize=2,
    linestyle='-', linewidth=0.2, color='r', label='Markov')

    ax.set_xscale('log')
    ax.set_xlabel('c values (logarithmic scale)')
    ax.set_ylabel('Maximum value of ' + r'$\rho$')
    ax.set_title('Maximum value of '+r'$\rho$'+' for which
    consistency still holds')
    ax.grid(True, which='both', linestyle='--', linewidth=0.2)
    ax.set_xticks([1, 2, 4, 10, 30, 60, 100])
    ax.set_xticklabels([1, 2, 4, 10, 30, 60, 100])
    ax.legend()

    plt.draw()
```

Now compute the maximum ρ values (in the code referred as x) for both *f1* and *f2* for all c values in their respective array. It then clears the axis and plot the updated data, sets a logarithmic scale for the x-axis to better visualize a wide range of values and execute this section every time the σ slider is updated.

```
fig, ax = plt.subplots()
plt.subplots_adjust(bottom=0.35)

ax_slider_s = plt.axes([0.25, 0.05, 0.5, 0.03])
```

```
slider_s = Slider(ax_slider_s, r'$\sigma$', 0, 1, valinit=0,
valstep=0.01)

update(0)

slider_s.on_changed(update)

plt.figtext(0.5, 0.18, r'$f1(\rho, c, \sigma) =\left(1 - \frac{2 (1 -
\sigma - \rho)}{c}\right) (1 - \sigma - \rho) - \rho$', ha='center',
color='blue', fontsize=10)
plt.figtext(0.5, 0.12, r'$f2(\rho, c, \sigma) =(1 - \sigma - \rho)
e^{-\frac{2 (1 - \sigma - \rho)}{c}} - \rho$', ha='center',
color='red', fontsize=10)

plt.show()
```

Finally it creates the initial plot and makes adjustments to the layout of the output window to make space for the slider. The slider is initialized with the value 0. The script connects the slider to the update function to dynamically update the plot and adds text labels for the functions.

B Best attack model solver

Here you can consult the commented code of the script used to solve the equations on the fork sustain attack analysis. The code can be consulted in its totality and without comments in the following GitHub repository: Videars/Blockchain-Consistency[2].

```
import numpy as np
import math
import matplotlib.pyplot as plt
```

We start by importing the libraries needed to run the script. In this case we need to handle a few vectors, perform basic mathematical operations and graph the results so *Numpy*, *Math* and *Mathplotlib* will be enough.

```
def clear_memo():
    global memo_P, memo_Delta
    memo_P = {}
    memo_Delta = {}

def calculate_Delta(k, s):
    if (k, s) in memo_Delta:
        return memo_Delta[(k, s)]

    if s - 1 >= k:
        result = 1
```

[2]https://github.com/Videars/Blockchain-Consistency

```
    elif s == -1:
        result = 0
    else:
        result = union*calculate_P(k-1, s)+h_delta*
        calculate_Delta(k-1, s-1)+delta_silent*calculate_P(k-1, s-1)

    memo_Delta[(k, s)] = result
    return result

def calculate_P(k, s):
    if (k, s) in memo_P:
        return memo_P[(k, s)]

    if s >= k:
        result = 1
    elif s == -1:
        result = 0
    else:
        delta_state = calculate_Delta(k, s)
        result = h * delta_state + a * calculate_P(k, s + 1)

    memo_P[(k, s)] = result
    return result
```

At this point we can define the model equations as global functions that can be called in the rest of the script. The functions, being recursive functions, will call themselves in the definition and to make the computation of solutions faster, we will use the memoization technique. This makes the script infinitely faster at the expense of more memory intensive use since we are going to store the already computed values of $P_s(k)$ in a dictionary so that we do not have to recompute in the next iterations. The dictionary must be reset when new parameters are defined so the *clear_memo* function is needed.

```
c_values = [1, 4, 60]
sleepy_fraction = 0.1
corrupts = [0.25, 0.40]
k_values = np.arange(2, 16, 2)

color_map = {1: 'purple', 4: 'orange', 60: 'green'}
plt.figure(figsize=(10, 6))
```

Next we define the 3 values of c for which we will compute the solutions and associate each of them with a color in the graph; this will allow the construction of a clearer graph to be consulted. Similarly, we define the fraction of corrupted nodes and all the values of k for which we will solve $P_s(k)$.

```
for corrupt in corrupts:
    if corrupt == 0.25:
        line_style = '--'
        adv_label = "25% Adv"
    else:
        line_style = '-'
        adv_label = "40% Adv"

    for c in c_values:
        corrupt_fraction = corrupt
        honest_fraction = 1 - sleepy_fraction - corrupt_fraction
        h = honest_fraction
        a = corrupt_fraction

        delta_silent = math.exp(-1/c)

        h_delta = 1 - math.exp(-(honest_fraction/2) / c)
        a_delta = 1 - math.exp(-corrupt_fraction / c)

        union = 1 - math.exp(-(corrupt_fraction +
        (honest_fraction/2)) / c)

        clear_memo()
        results = []
        for k in k_values:
            result = calculate_P(k, 0)
            results.append(result)

        plt.plot(k_values, results, line_style, label=f'{adv_label}'
        if c == c_values[0] else "", color=color_map[c])

from matplotlib.lines import Line2D
custom_lines = [
    Line2D([0],[0],color='black',lw=2,linestyle='-',label='40% Adv'),
    Line2D([0],[0],color='black',lw=2,linestyle='--',label='25% Adv'),
    Line2D([0],[0],color='purple',lw=2,linestyle='-',label='c=1'),
    Line2D([0],[0],color='orange',lw=2,linestyle='-',label='c=4'),
    Line2D([0],[0],color='green',lw=2,linestyle='-',label='c=60')
]

plt.legend(handles=custom_lines, loc='best')
plt.xlabel(r'$k$')
plt.ylabel(r'$P_0(k)$')
plt.yscale('log')
plt.title('Probability of winning the fork sustain attack game')
plt.grid(True)
plt.show()
```

Now, for each fraction of adversarial nodes in the defined vector, we compute the remaining parameters of the model according to the definitions given in chapter 6.4 and, for all k defined in its vector, we perform the computation of $P_s(k)$. The results are saved and plotted on the graph. Each solution will be represented with the color and line type defined by the value of the parameters ρ and c from which it was determined.

```
P_max = 1e-6
T_values = {}

for corrupt in corrupts:
    for c in c_values:
        corrupt_fraction = corrupt
        honest_fraction = 1 - sleepy_fraction - corrupt_fraction
        h = honest_fraction
        a = corrupt_fraction

        delta_silent = math.exp(-1/c)

        h_delta = 1 - math.exp(-(honest_fraction/2) / c)
        a_delta = 1 - math.exp(-corrupt_fraction / c)

        union = 1 - math.exp(-(corrupt_fraction +
        (honest_fraction/2)) / c)

        clear_memo()

        T_found = False
        k = 2
        while not T_found:
            result = calculate_P(k, 0)
            if result < P_max:
                T_values[(corrupt, c)] = k
                T_found = True
            k += 1

print("Calculated T-consistency (k for P(k, 0) < P_max):")
for (corrupt, c), T in T_values.items():
    print(f"For a = {corrupt}, c = {c}: T = {T}")
```

In this last part of the script, we calculate the value of the T parameter of the consistency property. First of all, we define the threshold value of the probability for which the attack could still be feasible, in this case set to 10^{-6}, and we re-run the computation of the solutions of $P_s(k)$, increasing k each time, until we find the T for which $P_0(T) < P_{max}$.

References

[1] Joseph Bonneau, Andrew Miller, Jeremy Clark, Arvind Narayanan, Joshua A. Kroll, and Edward W. Felten. Sok: Research perspectives and challenges for bitcoin and cryptocurrencies. 2015 IEEE Symposium on Security and Privacy, 2015. https://ieeexplore.ieee.org/document/7163021.

[2] Vitalik Buterin et al. Ethereum white paper. GitHub repository, 2013. https://static.peng37.com/ethereum_whitepaper_laptop_3.pdf.

[3] Kai-Min Chung, Henry Lam, Zhenming Liu, and Michael Mitzenmacher. Chernoff-hoeffding bounds for markov chains: Generalized and simplified. Leibniz International Proceedings in Informatics, LIPIcs, 2012. https://doi.org/10.4230/LIPIcs.STACS.2012.124.

[4] Dwork Cynthia and Naor Moni. Pricing via processing or combatting junk mail. Advances in Cryptology - CRYPTO '92, 12th Annual International Cryptology Conference, Santa Barbara, California, USA, August 16-20, 1992, Proceedings, 1992. https://doi.org/10.1007/3-540-48071-4_10.

[5] Kostis Karantias, Aggelos Kiayias, and Dionysis Zindros. Proof of burn. Cryptology ePrint Archive, Paper 2019/1096, 2019. https://eprint.iacr.org/2019/1096.

[6] Lucianna Kiffer, Rajmohan Rajaraman, and Abhi Shelat. A better method to analyze blockchain consistency. Cryptology ePrint Archive, Paper 2022/601, 2022. https://eprint.iacr.org/2022/601.

[7] Leslie Lamport, Robert Shostak, and Marshall Pease. The byzantine generals problem. ACM Trans. Program. Lang. Syst. volume 4, 2002. https://doi.org/10.1145/357172.357176.

[8] M. Milutinovic, W. He, H. Wu, and M. Kanwal. Proof of luck: An efficient blockchain consensus protocol. SysTEX 2016: 1st Workshop on System Software for Trusted Execution, Trento, Article No. 2, 2016. https://doi.org/10.1145/3007788.3007790.

[9] Satoshi Nakamoto. Bitcoin: A peer-to-peer electronic cash system, 2009. http://www.bitcoin.org/bitcoin.pdf.

[10] Takeshi Ogawa, Hayato Kima, and Noriharu Miyaho. Proposal of proof-of-lucky-id(pol) to solve the problems of pow and pos. IEEE International Conference on Internet of Things (iThings) and IEEE Green Computing and Communications (GreenCom) and IEEE Cyber, Physical and Social Computing (CPSCom), 2018. https://doi.org/10.1109/Cybermatics_2018.2018.00215.

[11] Rafael Pass, Lior Seeman, and Abhi Shelat. Analysis of the blockchain protocol in asynchronous networks. Cryptology ePrint Archive, Paper 2016/454, 2016. https://eprint.iacr.org/2016/454.

[12] Rafael Pass and Elaine Shi. The sleepy model of consensus. Cryptology ePrint Archive, Paper 2016/918, 2016. https://eprint.iacr.org/2016/918.

[13] Sheikh Munir Skh Saad and Raja Zahilah Raja Mohd Radzi. Comparative review of the blockchain consensus algorithm between proof of stake (pos) and delegated proof of stake (dpos). International Journal of Innovative Computing, 2020. https://ijic.utm.my/index.php/ijic/article/view/272.

[14] May Salama, Ziad Hussein, and Sahar Hassan. Evolution of blockchain consensus algorithms: a review on the latest milestones of blockchain consensus algorithms, 2023. https://doi.org/10.1186/s42400-023-00163-y.

[15] J. Yusoff, Z. Mohamad, and Anuar M. A review: Consensus algorithms on blockchain. Journal of Computer and Communications, 10, 37-50, 2022. https://doi.org/10.4236/jcc.2022.109003.