SOME THEORETICAL PROBLEMS IN BLOCKCHAIN SECURITY AND EFFICIENCY

BY

SURYANARAYANA SANKAGIRI

DISSERTATION

Submitted in partial fulfillment of the requirements
for the degree of Doctor of Philosophy in Electrical and Computer Engineering
in the Graduate College of the
University of Illinois Urbana-Champaign, 2022

Urbana, Illinois

Doctoral Committee:

Professor Bruce Hajek, Chair
Professor Pramod Viswanath
Assistant Professor Andrew Miller
Assistant Professor Ling Ren
Professor David Tse, Stanford University

Abstract

A salient factor behind the success of Bitcoin and other cryptocurrencies is the security of its underlying consensus engine: the longest-chain protocol. Recent literature has given us an excellent theoretical understanding of this protocol's behavior in the synchronous (bounded communication delay) model. One principle that emerges from this work is that the protocol's security degrades as the bound on the communication delays increases. It is therefore natural to investigate further the protocol's robustness to network delays and how one might enhance it further. This thesis studies the security of the longest-chain protocol in network models that relax the synchronous assumption. In particular, we answer the following two questions in the affirmative:

1. *In a network with random, possibly unbounded delays, is the longest-chain protocol secure?* This work shows that the protocol is robust to sporadic, large delays, as is wont to happen in real-world networks. Moreover, it sheds light on the dual role of delays on security: they have both a local effect and a global effect.

2. *In a network with more adverse conditions, i.e., occasional periods with arbitrary delay, How can the longest-chain protocol be made secure?* This work designs a *finality gadget* with provable security properties that ensures the security of the longest-chain protocol even under arbitrary delay. It also reveals some nuances about the CAP theorem, a fundamental impossibility result in blockchains and distributed systems.

In addition, Bitcoin suffers from poor transaction throughput. This is a fundamental limitation of the longest-chain protocol. Layer-two solutions provide a means to offload most of the transactions from the blockchain, using the consensus mechanism only when disputes arise. Payment channel networks are a class of layer-two solutions that have already been deployed in practice. In this dissertation, we model these networks mathematically, explore the benefits of dynamic routing and flow control, and present a network protocol that jointly performs these actions to steer the network to an optimal operating point. The protocol is derived as a method of solving a network optimization problem. It makes use of channel prices to co-ordinate the actions of the nodes in a decentralized fashion.

To my teachers and mentors, for shaping me into who I am.

Acknowledgments

This dissertation is the culmination of my six-year long Ph.D. journey. I would like to take the opportunity to thank my many companions and guides along this journey.

First and foremost, I would like to express my deepest gratitude towards my advisor, Prof. Bruce Hajek. He has been a role model for me in many ways. In particular, I have learned from him the importance of thinking clearly from first principles, working through simple examples to understand a difficult concept, and not stopping until one has truly imbibed the ideas in one's research so as to think about them with a high degree of fluency. I would like to thank him for the tremendous patience he has shown in mentoring me in my research, gradually encouraging me to grow more independent. I hope to carry forward all his teachings into my future career.

Next, I would like to thank my Ph.D. committee members, Prof. Pramod Viswanath, Prof. Andrew Miller, Prof. Ling Ren, and Prof. David Tse (from Stanford). Discussions with them have made me think more critically about my own research, and this dissertation is all the better for it. Periodically, throughout my stay at Illinois, I have interacted with them and these interactions have helped shaped my perspective about blockchains. Perhaps most importantly, their own foundational and remarkable research on blockchains has been an inspiration for me to study this topic for my dissertation.

I would like to thank my collaborators over the years. Xuechao Wang, Prof. Sreeram Kannan (from University of Washington) and Prof. Pramod Viswanath have been my co-authors on the work on blockchain CAP theorem that features in this dissertation. In addition, I wrote a paper with my friend, Bolton Bailey, that does not feature here. Working with all of them has been a real pleasure. Getting stuck on problems from time-to-time, brainstorming on their solutions, and staying up long nights chasing deadlines leave me with fond memories. I would also like to thank my internship mentors at IBM Research, Bangalore, namely Kameshwaran Sampath and Danda Sai Koti Reddy. Working with them on a totally different flavor of problems was a great learning experience, and a joyful one too.

Along with research, coursework has been an integral part of my Ph.D. experience as well. I would like to thank all my teachers from all the courses I took. They were all excellent instructors, and I enjoyed each of my courses thoroughly. Many of them serve as ideals for me to aspire towards. I am also indebted to Prof. Olgica Milenkovic, Prof. Zhizhen Zhao, and Prof. R. Srikant for allowing me the opportunity to teach in their courses. The experience was invaluable for me.

The Coordinated Science Lab, where my office was throughout the six years, is a wonderful ground for making strong bonds of friendship. I have been extremely fortunate to befriend many fellow students here, seniors, juniors, and contemporaries, without whom my Ph.D. experience would have been rather forlorn. I am deeply indebted to the many seniors from whom I have received kind words of mentorship and I hope I may similarly mentor those who undertake their Ph.D. journey after me. I would particularly like to thank my labmates, Dr. Xiaohan Kang, Dr. Zeyu Zhou, and Taha Ameen; I have learned a lot from them over the years. I am also deeply grateful to my flatmates for their constant, merry companionship over the years: Krishnakant, Pranjal and Suraj. Outside of UIUC, I have been fortunate to be part of the Champaign County Audubon Society birding community, the Iyengar Yoga studio at Champaign-Urbana, and the Illinois Vipassana Meditation Center (Dhamma Pakasa) at Pecatonica. My experiences at each of these places, as well as my friends and mentors from there, have transformed me in a beautiful way.

Last but not the least, I would like to thank my family for their constant support. My parents, and indeed, my extended family, have always valued learning, which has been a strong motivating factor for me to pursue my Ph.D. Finally, I would like to thank my wife, Vaishnavi Subramanian, for being there always, to provide support and encouragement through the tough times as well as celebrate the good times.

Table of Contents

Chapter 1

Introduction

1.1 Blockchains: A New Paradigm of Decentralized Internet Services

The internet is a modern engineering marvel that has influenced many spheres of our lives. At its core, it is a medium for exchanging digital information across devices in a seamless fashion. A salient feature of the internet is that it is, by design, a decentralized system. This means that there is no authority controlling what information gets shared on the internet and who has access to information. For example, any individual (or corporation/entity) is free to host a webpage with whatever content they please on the World Wide Web, provided they have secured rights to the domain name. Similarly, any user (or client) is free to access the content on any such webpage, at whatever time and frequency they choose to, provided they meet the appropriate terms and conditions set by the webpage, e.g., creating an account or paying a fee. Arguably, the decentralized aspect of the internet has allowed many creative enterprises to flourish around it.

Since the early 2010s, an increasing fraction of the internet usage has revolved around web-based platforms. For example, consider the case of hotel bookings. In principle, each hotel is free to host a webpage of its own, offering rooms to any customer willing to pay the price. Users are free to access webpages of any hotel they know of. However, it is more convenient for users to access all possible options via single platform such as Hotels.com or Expedia, where different lodging options are offered by the individual hotels themselves. The success of web-based platforms can be seen from the fact that they are found across various domains. Some examples of popular web-based platforms are Uber for ride-seeking, AirBnB for home-based lodging, Amazon for e-commerce, Spotify for music, etc.

The growing influence of web-based platforms has actually led to the centralization of control over the internet, especially with regard to financial transactions that take place over it. For example, consider the case of web search, the avenue through which most webpages are accessed. As of 2022, an overwhelming fraction of web searches are made through Google (see here); this has been the case for the past decade. A search engine company has the power to tweak its algorithms in a non-transparent manner, promoting some pages and suppressing others. Thus, a search engine behemoth can unilaterally influence the information that a large fraction of the population has access to. A monopoly in this space also means that the company has the power to monetize search results arbitrarily, leading to high prices for clients. The same principle carries over to more

specific platforms, too. For example, platforms like Amazon, AirBnB, Uber, Spotify, etc. can claim a large fraction of the transaction value between service provider and consumer, simply because they control a large share of the market in their respective segment. Broadly speaking, in today's world dominated by internet behemoths, service/content providers (e.g., drivers, artists, etc.) have limited control and ownership of their product. Thus, the internet world is not a very democratic one.

What are the key features that a decentralized internet platform should have? For one, the platform should be operated by a collection of individual entities (say, stakeholders) rather than one single entity. The rules of the operation, the data used by the platform, should be accessible to all. Any change in the operations of the platform would have to be agreed upon by all, or at least a majority of the stakeholders. The system should be fair: all parties should have roughly the same responsibility in maintaining the system, accruing benefits in proportion to the effort (say, storage/computation) they put into the system. For it to be truly decentralized, parties should be allowed to enter and leave the system at will.

Such a decentralized platform would necessarily be a distributed system. Indeed, stakeholders (or rather, their computers) who are geographically distributed need to communicate and cooperate to provide a seamless service to clients. Today's web-based platforms are also powered by a distributed system of servers; not all the data is stored and processed on a single computer. This is done for greater fault-tolerance and ease of scaling. However, despite being distributed, these systems are not decentralized. This is because all the computing nodes are controlled by a single agent and they can be programmed to run according to pre-specified rules. There is no notion of nodes acting of their own free will. In contrast, in a decentralized system, the participating nodes are independent entities. Thus, the prescribed protocol should incentivize nodes to correctly follow the prescribed protocol, and punish users for deviating from it (or make it incredibly hard to do so). The protocol must also be robust to arbitrary behaviors of the participating nodes and a faulty communication process. In particular, it should be hard for any one participating node to take control of the system.

The first such independent, decentralized, internet system that addressed the above challenges was Bitcoin, which started its operation in 2009. Bitcoin is a decentralized monetary transaction platform. It provides the service of a bank without requiring a single trusted party to monitor the transactions. Rather, the role of the bank is played by the participants at large. Any user is free to join Bitcoin as an active participant in keeping records of the transactions made in an ever-growing ledger, and ensuring that these transactions are not fraudulent. There are two salient aspects in such a decentralized system that differentiate it from a centralized banking system. First, the rules of the system are transparent and publicly known; in particular, users and observers know what constitutes a valid transaction as well as the monetary policy (when more money is injected in the system). Thus, in such a system, a client's assets cannot be arbitrarily frozen, nor can the liquidity in the system be arbitrarily changed. Second, the cost of making a transaction gets decided in an open market, instead of having a transaction fee set unilaterally by the bank/credit-card company.

Thus, such a system is fair from an economic point-of-view.

A key feature in the design of Bitcoin is the blockchain data-structure, or more simply, the *blockchain*. The blockchain is the ledger in which all transactions are recorded. At any given point in time, there are a set of active ledger maintainers in the system, called *miners*. These miners store a copy of the blockchain in their local memory; moreover, they make this ledger publicly accessible, which is essential to have a decentralized system. To extend the ledger, a miner creates a block (some reasonable number) of new transactions, appends this new block to the existing blockchain ledger, and then broadcasts this new block to the other miners. Miners take turns in adding new blocks to the blockchain, thereby streamlining the process. As the ledger is publicly verifiable, honest participants (miners and clients) can simply ignore those miners that add faulty blocks to the ledger. Security measures are also put in place to ensure that each miner has a fair chance at extending the ledger. Thus, no one party can censor transactions from reaching the ledger. In addition, miners are incentivized to remain active and to follow the protocol. We elaborate on the design of Bitcoin in Section 1.2 below.

Soon after the advent of Bitcoin, it materialized that such a publicly verifiable decentralized ledger could serve as the backbone for a much wider variety of services. Account balances could be replaced by more complex state information and transactions could be replaced by more complex programs that operate on the state variables and record changes in them. Thus, a blockchain-based system could actually power a decentralized computer that could perform much more complex tasks. Founded in 2013, Ethereum was the first system to implement this idea in practice. It allows clients to host their own programs (called *smart contracts*) on a decentralized computer (called *Ethereum Virtual Machine*). Moreover, the decentralized mature of the platform meant that clients have a greater sense of ownership of their smart contracts; they could tweak it whenever they pleased and charge money for other clients to use their smart contract according to whatever policy they decided. The sole job of the miners is to co-operate and maintain the basic infrastructure of the ledger. Moreover, their remuneration is decided by an open market where the miners are the sellers and the clients are the buyers.

The basic idea of a blockchain, introduced in the design of Bitcoin, is now used as a building block of many decentralized electronic transaction systems, called cryptocurrencies. Moreover, the study of blockchain-based systems has blossomed into an active area of research today. The major thrust of the research and development is on two fronts: building new decentralized systems using blockchains and improving the fundamental properties of a blockchain. The vision for blockchain-based systems is that they could be pivotal in scenarios where trusted third party is needed for data-exchange, monitoring, making negotiations, and enforcing regulations. For example, supply chain management often requires coordination and regulation among various independent agencies across borders, which could be cumbersome. A blockchain-based platform for supply chain management could ease these processes. On the more fundamental front, the major focus has been on improving the privacy guarantees and the transaction processing capacity of blockchains. Improvement on both these fronts is essential for blockchains to be adopted more widely.

We now turn to describe the design of Bitcoin in more detail. This description provides a context for stating the main contributions of this dissertation.

1.2 Bitcoin: The First Blockchain System

In 2008, Satoshi Nakamoto released the design of Bitcoin in a whitepaper titled "Bitcoin: A Peer-to-Peer Electronic Cash System" [1]. A few months later, on 3rd January 2009, the Bitcoin system officially began to operate. As the title of the paper suggests, Bitcoin provides an internet-based monetary transaction service. From a client's (i.e., user's) perspective, the service provided by Bitcoin is comparable to online banking or using apps such as PayPal. However, unlike traditional electronic payment systems, Bitcoin is a decentralized, peer-to-peer system. This means that there is no single agent (or entity) that controls or owns Bitcoin. Any person or entity could contribute in maintaining this electronic transaction service, joining and leaving whenever they pleased. Thus, a group of peers plays the same role as a bank or a credit card company would in a traditional transaction-processing system.

We first highlight the components of Bitcoin that are similar to any other electronic transaction system. Our description here is highly simplified. For a more comprehensive description, we refer the reader to the textbook by Narayanan et al. [2]. One salient feature of Bitcoin, which is common in nearly all electronic transaction systems, is the usage of digital signatures (See any introductory cryptography textbook, e.g., [3], for a description of digital signatures). Just like physical signatures, their digital counterpart allow any verifier to authenticate the source of messages. In Bitcoin, a transaction is a message indicating the transfer of money from the payer to the payee. Transactions are always signed by the the payer. Digital signatures cannot be forged. Thus, Bitcoin offers the basic security property that a client cannot spend (or steal) another's client money from their account.

A second salient feature of Bitcoin that is shared by all (electronic) transaction systems is that all transactions are recorded into a ledger. A ledger is simply an ordered list of pertinent items; these could be transactions, events, etc. As new transactions are made, they are continuously added to the ledger. The ledger helps keep track of the state of the system, i.e., the balances of (funds held by) the different users. Based on the state, a transaction can be deemed to be valid or not, in the sense, whether the payer has sufficient funds to make the transaction. Thus, a secure, tamper-proof ledger provides another basic guarantee: that a client's money does not disappear from the system.

We now take a look at the key factors differentiating Bitcoin from other electronic transaction systems. In a centralized system, the ledger resides on the servers of the controlling authority. Transactions are sent by clients to this central authority, which then verifies that these transactions are valid before adding them to the ledger. The entire ledger is not publicly accessible; a client can only see their own account details. In a decentralized system like Bitcoin, all miners store a copy

of the ledger in their local memory (recall that a miner is a peer maintaining the ledger). Clients send their transactions to some of the peers, who in turn broadcast the transaction to all of the other peers via a gossip protocol [2]. Each miner then individually validates transactions before considering them to be added into the ledger. Finally, nodes add new transactions to their ledger by participating in the *longest-chain protocol*.

To underscore the role of the longest-chain protocol, we take a look at two simple scenarios where such a protocol is not used. First, consider what would happen if miners added transactions to their ledger immediately upon hearing of them for the first time. Since every peer hears of different transactions at different times, the order of transactions could vary from one peer's ledger to another. Next, consider an alternative system where one peer decides on the list of transactions and communicates this ordered list to others. This one party would have complete control of writing into the ledger; in particular, they could censor out certain transactions according to their whims. Thus, this option is not a good one as well. Clearly, we need a protocol where each party has a fair chance of adding transactions to the ledger. Moreover, every party must agree on the order of transactions. Such a protocol is called a *consensus protocol*, of which the longest-chain protocol is an example. We describe this protocol in the following praragraphs.

Let us imagine, for the moment, that the Bitcoin system has a leader election oracle, which periodically selects one miner among the set of miners as a leader. The leaders are chosen at random and each leader is chosen independent of the past. Once a miner is elected as a leader, it creates a new block of transactions; here, a block is an ordered list of a moderate number (around a thousand) of transactions. Only those transactions that the leader has heard of, but are not yet recorded in the ledger, are included in the new block. Having created this new block, the leader appends this block to the end of its ledger. These blocks, when chained together, one after the other, naturally form a ledger of transactions. The term blockchain, which today has grown to represent the whole system, was originally used to refer to this particular linked-list-like representation of a ledger. Lastly, the leader broadcasts this block to all other miners. Upon hearing of the new block, each miner individually validates the block and includes it in their ledger. Grouping the transactions into blocks helps streamline the ledger-update process.

In Bitcoin, the leader election oracle is implemented through a clever use of the *proof-of-work* idea. The term proof-of-work (or PoW) arises in the context of solving hash puzzles, or more generally, computational problems that are hard to solve but easy to verify. These hash puzzles are built using cryptographically secure hash functions, which are computationally hard to invert. By extension, this means that it is also hard to find an input whose output string meets some particular condition. In Bitcoin, the hash puzzle is to find an input whose hash is less than a certain threshold. Equivalently, the bit string representation of the output must begin with a certain number of zeroes (say, twenty). The best (and effectively, only) method of solving the hash puzzle would be to sample numbers at random and compute this random number's hash. On average, after 2^{20} such attempts, one can find an input whose hash meets the aforementioned criterion. Thus, performing this task requires a fair amount of computational *work*. Crucially, a

party that has performed this computational task can provide to a verifier party this hard-to-find input. This input is called the proof-of-work. The verifier can compute its hash (fairly quickly) and verify that the original party has indeed performed a fair amount of computational work.

In Bitcoin, miners are constantly competing with each other, trying to solve a particular hash puzzle the fastest. The first miner to solve the hash puzzle is considered the leader and has the right to propose a new block. The lucky miner includes the proof-of-work along with the block and broadcasts it to other miners. Those receiving a block can verify that the proof-of-work is valid, and are therefore assured that the leader is bonafide. Note that the miner who solves the hash puzzle first is chosen at random, since each miner is trying to solve the hash puzzle independently. Thus, the proof-of-work mechanism gives rise to a decentralized, randomized leader election mechanism.

The precise puzzle that the miners are trying to solve is to create a block whose hash is less than a certain threshold. Among the contents of the block is a special field called the nonce, which is exclusively meant to be a parameter that can be varied to adjust the hash. The hash puzzle for every new block is a fresh puzzle. Therefore, its winner is independent of the previous hash puzzles. The term miner itself originates in this proof-of-work setting: those seeking to solve the hash puzzles are mining for a precious resource (a valid nonce) by spending valuable (computational) energy.

We summarize our description of Bitcoin so far in the following points.

1. New transactions are continuously generated by clients in the system, and are broadcast to the miners.

2. Miners collect all new transactions that they have heard of, but are not yet included in the ledger, into a block.

3. Miners continuously work on finding an appropriate nonce such that the hash of their entire block is less than a certain threshold. Upon finding such a nonce, they broadcast the block to all other miners.

4. Miners accept a new block only if all transactions in it are valid and the block also includes a valid nonce. Once they accept a block, they begin working on creating the next block in the chain.

We now examine some aspects of the protocol's robustness and security. For one, each transaction that is made must be signed by the payer. Thus, no user can spend any other user's money. Suppose a user tries to cheat by creating a transaction that spends more money than they own. Such a transaction can easily be caught by the miners when they validate the transaction. In fact, no honest miner would include such a transaction in the block they are mining on. Suppose a corrupt miner does include such a transaction in its block and produces a valid proof-of-work for it as well, all other miners would ignore such a block. Thus, fraudulent transactions are easily discarded from the ledger.

The protocol is robust against communication delays as well. For instance, at any given point in time, the new block that miners are mining on possibly differs from one party to another. This is

because a transaction may have made its way to some miners but not to others. However, as long as the transaction is valid, it is bound to be recorded in the ledger eventually. The protocol also accounts for the fact that there are communication delays in sending a block from one miner to the others. Suppose, before a certain newly mined block reaches others, another miner is successful in finding a nonce and broadcasts its block. These two blocks will both be pointing to the same parent block. Sooner or later, all miners will hear of both these blocks. Miners treat the block with the lower hash as the valid block and continue building on it; the other block is simply ignored. More generally, due to network delays, it is possible that there are *forks* in the blockchain. The rule that miners follow is to pick the chain with the largest amount of 'work'. This essentially means picking the longest chain, and breaking ties among equally long chains by choosing the one with the smallest sum of hashes. This rule gives the longest-chain protocol its name. The longest-chain rule is also applicable for any party who wants to merely view the ledger or perhaps wants to newly get started with mining. As such, such a user could be presented with many different chains. The longest-chain rule guides the choice of the chain in this case. It also resolves the dilemma of what is the authentic copy of the blockchain.

In our description of Bitcoin, we have omitted a description of the monetary policy of Bitcoin (how new coins are introduced in the system), as well as the incentives for miners to participate in the system. These aspects, although very important, are somewhat orthogonal to the focus of this dissertation, which is on the security and transaction-processing efficiency of blockchains. We refer the reader to the book "Bitcoin and Cryptocurrency Technologies" by Narayanan et al. [2] for a detailed discussion on these aspects.

1.3 The Longest-Chain Protocol: Security and Efficiency

1.3.1 Security From a Consensus Viewpoint

Arguably, the most important problem of study in the Bitcoin system is that of security. Assuming that standard cryptographic constructs are secure (and are correctly implemented), the question of security boils down to whether the longest-chain protocol truly provides consensus. To elaborate, the question of interest here is to under what conditions does the Bitcoin system provide one consistent view of the blockchain to all the miners and the clients in the system. We illustrate the importance of having consensus among the miners by describing what is known as the private attack. In this attack, an adversary aims to obtain some real-world good without actually paying for it. In the example that follows, the adversary cheats a car dealer by obtaining a car without paying for it. Therefore, this attack is also known as the *double-spend attack*.

The sequence of events in this attack are as follows. First, the adversary issues a transaction `Tx1` paying a Bitcoin to a car dealer in order to buy a car. The transaction is legitimate, and so is picked up by some honest miner and included in a block B_1 which becomes part of the main blockchain.

Once this happens, the car dealer assumes that the transaction is permanently recorded on the ledger and hands over the car. Meanwhile, the adversary creates a second transaction `Tx2` that pays the same Bitcoin to itself, but to a different account. It does not broadcast `Tx2`; rather, it includes this transaction in a new block B_2 and tries to find a valid proof-of-work for this block. B_2 is purposely created to be in a chain parallel to B_1; i.e., they both have the same parent.

The adversary continues to work privately on building a chain of blocks that extend B_2. The honest miners, unaware of block B_2 and its descendants, continue extending the chain containing B_1. Suppose a moment comes where the adversary's privately held chain grows longer than the chain held by the honest miners. At such a moment, the adversary broadcasts its privately held chain to all other miners. All honest miners, following the longest-chain rule, adopt the chain sent by the adversary, which contains block B_2, and continue extending it. Block B_1 thus falls off the longest chain. The transaction paying the car dealer is no longer recorded on the ledger. Moreover, this transaction will never be included below block B_2 as it is now an invalid transaction; the coin in question is already spent.

With careful consideration, we see that the security threat stems from a lack of consensus. Indeed, the attack is successful because at some point in time, honest miners are presented with a chain that is viable (i.e., long enough) and differs substantially from the one they held until that point. More succinctly, there were two equally long chains among the peers that forked back quite some distance. Roughly speaking, the presence of multiple equally long chains implies that the miners cannot agree on any one chain by following the longest-chain rule. Thus, for the protocol to be secure, there must be one unanimous longest chain in the system at all times. Moreover, the longest chain at any time must be an extension of the previous longest chain. In other words, the longest chain must be strictly growing with time. The early works analyzing blockchain security, e.g., Garay et al. [4], Sompolinsky and Zohar [5], and Pass et al. [6], clearly make the connection between security of the underlying consensus mechanism and the security of the blockchain as a whole.

Whether the adversary is able to successfully execute a double-spend attack depends on whether it is able to privately build a chain that eventually grows longer than the honest miners' chain. This, in turn, depends on the fraction of mining power the adversary controls, i.e., on the computational resources the adversary can deploy towards this attack. Intuitively, if the adversary controls more than half the mining power in the system, then it can mine blocks at a faster rate than all the honest miners put together. In such a scenario, the adversary would be able to easily execute the attack. Conversely, if the adversary controls less than half of the computational resources, it would be hard to execute such an attack. Thus, the system is secure as long as the computational power of the adversary remains appropriately bounded. Assuming that there are many miners in the system with considerable computational power each, it is unlikely that a malicious party can muster the resources that exceed that of all the honest parties combined. This argument is the crux of the security properties of the longest-chain protocol.

In the original Bitcoin whitepaper [1] itself, Nakamoto made the following important securit

observation. Even with a small fraction of the mining power (say, ten percent), it is possible for the adversary to carry out a short-range private attack. This means that the adversary can create a short private chain, say of three blocks, which is momentarily longer than the honest miners' chain. This happens purely as a matter of chance; the adversary could get lucky and mine three blocks in quick succession. In this case, the private attack would eliminate two honest blocks. However, by incurring some latency, clients can guard against being cheated. With respect to the example above, suppose the car-dealer decides to wait for five more blocks to be built on top of B_1 before treating the transaction paying itself as confirmed. Before the transaction is confirmed, it would not hand over the car to the adversary. Thus, even if the adversary performs a short-range private attack, the net effect would be that the car dealer and the adversary did not make any transaction at all. However, if the adversary is able to build a chain of six blocks or more, and moreover, at some point, have it longer than the honest miners' chain, then it would be able to successfully carry out the attack.

The Bitcoin whitepaper [1] showed that for as long as the adversary has less than half the mining power, the probability with which it would be able to carry out a private attack overturning k or more blocks decreases exponentially with k. Smaller the adversarial fraction, the faster the decay rate in the adversary's chance of success. Thus, clients can get progressively better security guarantees by waiting for their transactions to be buried deeper before confirming them. However, this wait increases the latency of the transaction. Thus, the longest-chain protocol suffers from a latency-security trade-off.

1.3.2 Security Results: The Effect of Network Delay

Nakamoto's calculations were based on the assumption that all blocks are delivered instantaneously from the successful miner to all other miners. Under such an assumption, each honest miner hears of the block mined by the previous honest miner. Thus, the later miner creates a block extending the previous one's chain. Thus, the chain held by honest miners grows at the same rate as the creation of new blocks in the system. On the other hand, when there are network delays, forks emerge, and the honest miners' chain growth slows down. This is because network delays cause honest blocks to be built in parallel to each other, which means that not all blocks contribute to the blockchain's growth. This drop in the growth rate gives the adversary an advantage in the private attack. To be successful, the adversary only needs to mine a chain longer than the longest of the honest miners' chain, which is smaller than the number of honest blocks mined. The larger the delays, the slower is the growth rate of the honest miners, and the easier it is for the adversary to break consensus. Thus, security guarantees decrease with network delay.

In addition to reducing the growth rate of the honest chain, network delays are be detrimental to consensus in other ways as well. For example, due to delays, it is possible that an honest miner has not heard of the last few honest blocks. Thus, its chain is shorter than the longest chain in the system. This makes it easier for the adversary to attack such a miner. In fact, in the presence of

network delays, the adversary can do better than simply mine blocks in private and releasing them all at once. It can exploit the forks in the chain naturally caused by delays to propagate the fork it is trying to create. Put differently, the adversary can use some honest blocks to its own benefit. Accounting for all these factors makes it challenging to analyze the security of the protocol under network delays.

Over the years, a number of works in the literature have taken steps towards sharply characterizing the protocol's security properties under network delays. Nearly all of these works assume the synchronous communication model. Under this model, all messages among honest miners are delayed by no more than some known constant Δ. Beyond this, the adversary has complete control of the message delays. All works provide security guarantees of a similar flavor, which can be described as follows. For any given delay bound Δ and mining rate f, let the fraction of adversarial mining power be less than some $\beta(f\Delta)$. Then the probability with which a block buried k-deep in the longest chain ever gets removed from the longest chain decreases exponentially with k. A similar guarantee can also be given for blocks in the longest chain that were created at least k time units ago. These guarantees hold independent of the actions of the adversary, i.e., they hold irrespective of where the adversary chooses to mine and when and to whom it chooses to send its blocks. The term $\beta(f\Delta)$, the maximum tolerable adversarial fraction given the system's parameters, is called the *security threshold*.

The security threshold obtained in initial works [4, 6] were not tight. From simple calculations in [5], it was known that the private attack would be unsuccessful (with high probability) if $\beta < (1-\beta)/(1+(1-\beta)f\Delta)$. However, security was proven to hold only for a smaller value of β. In 2020, two sets of reserachers, namely Dembo et al. [7] and Gaži et al. [8], proved that the longest-chain protocol is secure if and only if *beta* is less than the private attack threshold. Thus, these works showed that asymptotically, the private attack was as good as any other attack. Since these works focused on the security threshold, their security guarantees were given in terms of order notation $(\exp(-\Omega(k)))$. These guarantees did not translate into pracitcal latency bounds for clients with a given tolerance for failure probabilities. The work of Li et al. [9] sought to address this gap. Recently Gaži et al. [10] provided practical finite-time security guarantees that are very close to lower bounds, and are thus nearly optimal.

1.3.3 Efficiency Limitations and Layer-Two Solutions

The security analysis of the longest-chain protocol reveals that the protocol has a very low *transaction throughput*. The transaction throughput is defined as the number of new transactions the system is able to process per unit time. The throughput of the protocol is given by the product of the number of blocks mined per unit time, i.e., the mining rate, and the number of transactions per block. Indeed, the throughput of Bitcoin is around five to ten transactions per second. In comparison, traditional payment systems like Visa or Mastercard process tens of thousands of transactions per second. The poor throughput of the longest-chain protocol has hindered the widespread adop-

tion of Bitcoin and other blockchain-based systems for payments. Naively, the throughput of the protocol could be improved by increasing the mining rate, the number of transactions per block, or both. However, increasing either of these parameters compromises on security. The security results mentioned in the previous section tell us that the security threshold decreases with the mining rate for a fixed network delay Δ. Thus, increasing the mining rate is ruled out. Increasing the block size leads to an increase in the communication delays, which also adversely affects security. Thus, the limited throughput is a fundamental drawback of the longest-chain protocol.

In recent years, the blockchain community has proposed a variety of methods to address the challenge of improving a blockchain's throughput. The proposed improvements can be broadly classified into two distinct categories: *layer-one solutions* and *layer-two solutions*. Layer-one solutions propose a new consensus mechanism that does not suffer the aforementioned drawbacks of the Nakamoto consensus protocol. In contrast, layer-two solutions use the existing framework of a Nakamoto consensus-based blockchain to provide users a tool to transact without recording all of the transactions on the blockchain itself. Hence, they are also termed as *off-chain mechanisms*. By reducing the transaction load on the underlying blockchain, layer-two solutions address the issue of low throughput. Further, some mechanisms have the following property: when both nodes act honestly, the mechanism provides a faster transaction confirmation guarantee than the consensus layer. Only when there are disputes, the transactions are broadcast to the whole network, and the history as recorded by the blockchain is treated as final. The recent survey paper by Gudgeon et al. [11] gives an overview of popular layer-two scalability solutions. They say that "the promise of layer-two protocols is to complete off-chain transactions in sub-seconds rather than minutes or hours while retaining asset security, reducing fees and allowing blockchains to scale" [11].

1.4 Contributions of This Dissertation

This dissertation studies three theoretical problems in the space of blockchains. The first two problems concern the security of the longest-chain protocol, especially under atypically large communication delays. In Chapter 2, we extend the security results for the longest-chain protocol from the constant delay setting to a network setting where delays are random and potentially unbounded. Our work illustrates how the proof techniques from the constant delay setting can be generalized to handle random delays. Next, in Chapter 3, we present a consensus protocol that can be viewed as augmenting the longest-chain protocol with a finality gadget. This gadget secures the consensus protocol under arbitrary network delays. The last chapter, Chapter 4 focuses on payment channel networks, a tool to improve the transaction throughput of blockchains. We show that such networks cannot serve arbitrary transaction requests over a long time. We then present a protocol that meets as large a fraction of the transaction requests made as possible.

1.4.1 The Longest-Chain Protocol Under Random Delays

In Section 1.2, we established that network delays play an important role in the security of the longest-chain protocol. For a given mining rate, larger the network delays, poorer the security threshold of the protocol. Consider a network where delays are typically small but are sporadically large. What would be the security threshold of the longest-chain protocol in such a network? Going by the chain-growth rate intuition, the threshold should be similar to what would be the case if all delays were the typical delay; perhaps a little worse. This is because if most blocks are delivered from one miner to the next with a small delay, then the growth rate of the chain would be nearly as high as the case if large delays never occurred. However, to apply the guarantees obtained from the analysis in the synchronous model, one would have to account for the worst-case delay. This would yield much poorer security guarantees than what the intuitive argument suggests. The reason for this is that although the synchronous network model allows for arbitrary delays within the prescribed delay upper bound, the security analysis considers the worst-case scenario, which is essentially the case that all messages are delayed by the maximum possible value. Thus, security guarantees from the synchronous model do not capture the distribution of delays within the prescribed range. In particular, these guarantees may be too pessimistic if the delay distribution has a large weight far lesser than the upper bound. A pessimistic security guarantee also translates to a poor choice of the mining rate for the system, which can be detrimental to the blockchain's throughput.

In order to obtain better security guarantees under inhomogenous network delays, it is important to model the network accordingly and simultaneously, build a security analysis that explicitly accounts for these inhomogeneous delays. In this work, we model the network by assigning message delays to be random variables, that are independent and identically distributed. The distribution can be set to be equal to the network's delay statistics that can be empirically measured. This communication model is a strict generalization of the synchronous model. It also allows for delay distributions with unbounded support. Within this network model, we derive safety and liveness guarantees that hold for infinite-horizon executions. We provide simple, explicit bounds on the failure probabilities that decay exponentially with the security parameter. We analyze the protocol in both the proof-of-work and proof-of-stake variants. Thus, our security guarantees are of the same flavor as existing results in the synchronous network model. The main takeaway is that the theoretical results in this work reinforce a belief that has been borne out by practice: the longest-chain protocol has good security properties in real-world network conditions.

The main technical challenge in this work is to extend the proof technique from the synchronous setting to the more inhomogenous network setting. One of the key insights from earlier works that we retain is the intuition that the longest-chain produced by the honest miners must outgrow the longest-chain by the adversarial users. We note that in calculating the growth rate, the precise delays do not matter; the only event of significance is whether a new miner has heard of the block(s) currently at the tip of the blockchain or not. Moreover, in the network model we propose, there is

a certain renewal structure in the occurence of these events; in other words, successive events are independent of each other. Thus, the growth rate of the honest longest-chain is dictated by the probability of an honest block hearing of a previously mined block; this, in turn, depends on the relation between the typical delay and the mining rate. Moreover, we also account for the fact that different honest users may have different versions of the longest-chain; this is also a consequence of inhomogenous delays. Under the random-delay assumption, we show that the chain held by any one party is not shorter than the longest-chain by too large a factor with high probability. This guarantee allows us to make security guarantees similar to that in the synchronous model.

1.4.2 Finality Gadgets and the Blockchain CAP Theorem

The longest-chain protocol is not secure in network conditions where the delays could be arbitrarily large. Consider, for example, the case where the communication network suddenly develops a partition, splitting the parties into two halves that cannot communicate with each other. These two sets of parties will build two separate blockchains that grow in parallel which fork off from the point where the network was last connected. When the network heals after a considerable period of time, the parties will have to choose one of the two chains to extend forward; the parties that were building on the other chain lose out on safety.

As such, no consensus protocol can make progress in a secure way during periods of arbitrary network delays. Classical consensus protocols like PBFT [12] are designed to remain secure under such adverse network conditions essentially by stalling. This behavior is encoded within the protocol itself by means of a vote-counting step: the protocol does not make progress unless each party receives a certain number of votes from other parties. Under network partition, this condition is not met, so the protocol does not confirm new blocks. In contrast, the longest-chain network continues to make progress even if communication is disrupted. There is no inbuilt mechanism that allows for the protocol to stall if communication with other peers cannot be established.

However, it is this very property that works as an advantage for the longest-chain protocol in environments with a varying number of participants. In scenarios where some honest parties may temporarily drop out of the protocol (or go to sleep), the longest-chain protocol continues to make progress in a secure fashion, while other permissioned consensus protocols stall. The mechanisms that safeguard the protocol in times of network asynchrony prove to be a hindrance in a setting with sleepy users. Indeed, in a decentralized system, it is impossible for a single party to distinguish whether other parties have gone to sleep or are active but cut-off from communication; in both cases, communication between parties is lost. Thus, this dichotomy between choosing security in a partially synchronous environment and making progress even when some parties are sleepy is a fundamental one. This trade-off is captured by the blockchain CAP theorem [13]. To be precise, the theorem suggests that a blockchain protocol by design must choose between finality (offering a deterministic security guarantee in the presence of network partitions) and adaptivity (remaining live and available in a setting with variable number of users).

In Chapter 3, we propose a new blockchain protocol, called the checkpointed longest chain, that offers individual users the choice between finality and adaptivity instead of imposing it at a system-wide level. This protocol's salient feature is that it supports two distinct confirmation rules: one that guarantees adaptivity and the other finality. The more optimistic adaptive rule always confirms blocks that are marked as finalized by the more conservative rule and may possibly confirm more blocks during variable participation levels. Clients (users) make a local choice between the confirmation rules as per their personal preference, while miners follow a fixed block proposal rule that is consistent with both confirmation rules. The proposed protocol has the additional benefit of intrinsic validity: the finalized blocks always lie on a single blockchain, and therefore miners can attest to the validity of transactions while proposing blocks. Our protocol builds on the notion of a finality gadget, a popular technique for adding finality to longest-chain protocols.

1.4.3 Pricing and Routing in Payment Channel Networks

Juxtaposed with the grand vision of bringing about a decentralized internet is the limitation that Bitcoin cannot support more than tens of transactions (or instructions) per second. This limitation stems from the longest-chain protocol itself. The transaction throughput of the protocol is given by the product of the mining rate (new blocks per unit time) and the number of transactions that can fit into a block. The latter parameter influences the size of the block and, consequently, the communication delay in sending the block from one miner to another. To increase the throughput, one could increase either the mining rate or the size of the block, or both. However, any of these actions increases the chances that the a block cannot be communicated from its miner to the others before the next one is mined. An increase in the frequency of this event leads to poorer security guarantees for the protocol. Thus, any instantiation of the longest-chain protocol is bound to have limited throughput, which would not suffice for modern-day applications.

Recent blockchain literature has sought to address this limitation of blockchains via a myriad of proposed solutions. Broadly, the approaches can be categorized into two: layer-one solutions and layer-two solutions. Layer-one solutions seek to replace the Nakamoto consensus protocol with an altogether different consensus mechanism that does not suffer from these limitations, e.g., Prism [14]. Sharded blockchains are another broad class of scaling solutions, where different sets of parties process a different set of transactions. However, a drawback of these solutions is that they cannot be implemented on existing blockchain systems.

Layer-two solutions address this drawback by providing backward compatible scalability solutions. At a high level, layer-two solutions operate by offloading transactions from the blockchain ledger. At its initialization, a certain portion of the blockchain's state is locked by the layer-two mechanism by recording an appropriate transaction on the blockchain. Then, the parties involved can transact via a different protocol as dictated by the layer-two protocol. This protocol can typically process transactions much faster than the Nakamoto consensus protocol. Finally, the layer-two mechanism releases funds by recording the state at the culmination of all the transactions

that took place off the blockchain. Thus, only the net effect of multiple transactions is recorded on the blockchain, thereby effectively increasing the number of transactions it can process.

A popular layer-two scalability solution is that of a *payment channel*. It is an escrow fund between two parties, created by the deposit of funds by the involved parties into a special account. The two parties then transact by exchanging messages with each other. These messages note how the escrow fund shall be ultimately split between the transacting parties. Thus, each message effectively marks the transfer of funds from one party to another The blockchain is used only to create and close the channel, or to handle disputes. The functionality of a payment channel is significantly enhanced when many of them join to form a payment channel network (PCN). Here, any party can transact with any other party as long as they are connected by a path of payment channels. A PCN saves on the amount of collateral locked in escrow, apart from time and transaction fees. The Lightning Network, a popular PCN, powers many of the apps used to transact in Bitcoin [15].

For a payment channel network to function efficiently, it is vital to have a principled protocol to route transactions across it. Varma and Maguluri [16] show that a dynamic, state-dependent routing protocol can help the network serve all transactions up to its capacity. At the same time, they also show that for the network to meet all the demands, the demand structure must satisfy the following stringent balance criterion: the net demand out of each node equals the net demand into each node. If this condition is not satisfied, the network is bound to have a non-zero net flow of money on at least some channel in the network, eventually skewing the balances completely. When this happens, the imbalance-causing flows naturally become infeasible, and the network continues to serve only a portion of the demands that satisfy the balance criterion. However, it is possible that certain deadlocks arise: demands that could have been served by the network without altering channel balances are infeasible because of the skewed channels. This phenomenon was first studied by Sivaraman et al. [17].

Motivated by these considerations, we ask the question: is it possible to extend the protocol of Varma and Maguluri [16] so that transacting nodes also make flow-control decisions, i.e., hold back on their transactions if routing them are leading to a deadlock? In Chapter 4, we present a joint flow-control and routing protocol, answering this question in the affirmative. Under stationary demand conditions and with channels of sufficiently large capacity, the protocol serves as large a fraction of transactions requests as possible without skewing the balances of the channels. The protocol is derived in the following steps. We first state the objective of the network as a constrained optimization problem. Next, we derive the dual of this optimization problem. The protocol is then derived as a gradient-descent method to solve this dual problem. The Lagrange multipliers, i.e., the variables in the dual problem, play a key role in the protocol. They can be interpreted as prices that the channels quote to the transacting nodes for routing a flow through it. For each transaction, its source-destination pair makes routing and flow-control decisions by optimizing for their personal utility with the routing costs subtracted. The channels then update their prices in a manner such that ultimately, the network approaches the desired state. We illustrate the behavior of the protocol via simulations.

Chapter 2

The Longest-Chain Protocol Under Random Delays

2.1 Introduction

Blockchains are peer-to-peer systems that maintain an ever-growing ledger. The term blockchain refers to the linked-list-like data structure that forms the ledger in these systems. Over the past decade, blockchains have proved useful in enabling decentralized payment systems, i.e., cryptocurrencies. In all such systems, the peers (or parties) participate in a consensus protocol to ensure that they maintain consistent copies of the ledger. While the blockchain data structure itself has remained fairly standard, associated consensus protocols have proliferated (see [18, 19] for surveys on consensus protocols). The *longest-chain protocol*, also known as the Nakamoto consensus protocol, is one such consensus mechanism. It was proposed in the design of Bitcoin [1], and is used by many cryptocurrencies today.

The longest-chain protocol has a simple description. Periodically, a party is elected as a leader via a randomized mechanism. The mechanism may be Proof-of-Work mining, as in Bitcoin [1], or some Proof-of-Stake mechanism, as in Ouroboros [20, 21, 22]. Once elected, a leader creates a new block at the end of its blockchain and broadcasts the block to other parties, who receive the block after some delay. Each party builds upon the longest blockchain it has heard of thus far. The protocol ensures that no one peer can single-handedly manipulate the ledger.

A fundamental problem is to study the conditions under which the protocol is secure, despite the presence of malicious parties who can deviate from the protocol arbitrarily. Starting from the pioneering work of Garay et al. [4], many works have analyzed the protocol under a variety of modeling assumptions [6, 20, 23, 7, 8]. From this literature, some basic principles emerge that hold irrespective of the model.

One fundamental principle is that the protocol's security can be distilled down to two properties: a safety property, namely *settlement*, and a liveness property, namely *chain quality*. The settlement property (aka persistence, consistency or common prefix) states that blocks in the blockchain that were built more than some k time units ago become settled, i.e., remain in the blockchain forever after or remain out. The chain quality property states that every k time units, at least one block built by an honest leader will appear in the blockchain. In essence, settlement states that all parties have consistent blockchains and liveness states that they keep including new transactions. When both properties hold, the protocol is said to be secure.

A second principle is that the longest chain protocol is secure when the fraction of adversarial parties (more specifically, adversarial representation power) is below a certain threshold and is insecure otherwise. The threshold depends on the mining rate, i.e., the leader election rate f, and the network delays Δ. In the *synchronous network model*, a message sent at time τ will be delivered by time $\tau + \Delta$, where Δ is a known constant. In such a network, the protocol is secure if and only if

$$\beta < \frac{1-\beta}{1+(1-\beta)f\Delta}, \tag{2.1}$$

where β is the fraction of adversarial power [7, 8]. Moreover, if the tuple (β, f, Δ) satisfies (2.1), the probability that security is broken decreases exponentially with the wait time k [8, 10].

Note that the security threshold, i.e., the supremum of β for which the protocol is secure, decreases with $f\Delta$ (set the inequality in (2.1) to equality). In other words, network delays have a detrimental effect on security. This phenomenon can be understood as follows. If a leader does not hear of the most recent block, its block does not extend the blockchain and is wasted. If network delays are large, this happens frequently, and the rate at which the blockchain grows slows down. Effectively, the honest participation reduces, giving the adversary an advantage.

In a real-world network, the worst-case message delays may be much larger than typical delays (see, e.g., the empirical study by Decker and Wattenhofer [24]). Often, these are due to random factors such as network traffic or noise in the channels. In such a scenario, the only guarantees from the synchronous model are obtained by setting Δ to the maximum possible delay in (2.1). However, this may yield pessimistic guarantees: the fraction of honest blocks wasted would typically be less than that suggested by the worst-case delay. We posit that better security guarantees may be obtained by studying the protocol under a new communication model that allows for random delays.

2.1.1 Our Contributions

The main contribution of this chapter is to analyze the security of the longest-chain protocol in a network with random, possibly unbounded, delays. Briefly, the network model we consider is as follows. Each peer-to-peer communication is subject to an independent and identically distributed (i.i.d.) delay. Thus, different recipients of a broadcast may receive the same broadcast message at different times. This model generalizes the synchronous model and has not been studied in prior work on blockchain security. For this model, we obtain security results that are qualitatively similar to those in the synchronous model. For ease of exposition, this work focuses on the simpler Proof-of-Work setting instead of the more general Proof-of-Stake setting (see the work of Blum et al. [25] for a comparison between Proof-of-Work and Proof-of-Stake).

A secondary contribution of ours is that we explicitly state and distinguish between two forms of security properties: *intensive* and *extensive*. These terms are borrowed from statistical physics. Intensive properties capture the security of localized portions of blockchains, whereas extensive

properties provide global security guarantees. Prior works typically state properties in either one of these forms; those that state properties in both forms do not formally distinguish them. We show that guarantees for the intensive forms imply guarantees for the extensive forms.

Our main result, Theorem 2.1, states that the longest-chain protocol is secure in the random delay model, except with probability that decays exponentially in the wait-time (or security parameter) k. The condition for security is the following: the probability with which the miner of a block is honest *and* has heard the block broadcast by the previous miner must be greater than a half. Note that this condition allows for delay distributions with a long, possibly unbounded, tail. When delays are zero, this reduces to the honest majority condition. The result addresses the settlement and the chain quality property jointly. We provide simple, explicit error bounds. The guarantees hold for infinite-horizon executions.

On a finer note, our work shows that communication delays have both global and local effects. Delays from past leaders to future ones have a global effect: they influence the growth of the longest chain and impact the security of all honest parties. We incorporate these delays in our definition of the *characteristic string* (Section 2.4.2). In contrast, delays from leaders to a given honest party h have local impact: they affect the length of the chain held by h, and, therefore, h's security alone. We account for these delays in the term $\mathsf{Unheard}_h$ (Section 2.4.3). This dual role is reflected in Theorem 2.1, where the error bounds have two terms. One is a bound on atypical behavior of the characteristic string, while the other is a bound on atypical behavior of $\mathsf{Unheard}_h$ for every honest party h.

2.1.2 Comparison with Prior Work

Communication Model The random delay model of this chapter is a relaxation of the synchronous network model. There are two other relaxations that are popular in the literature: the *partially synchronous model* [26] and the *sleepy model* [23]. Both models relax the synchronous assumption by allowing the adversary to arbitrarily delay some messages. This adversarial power is localized–to select time periods in the partially synchronous model and to select parties in the sleepy model. More formally, the partially synchronous model assumes that message delays are unbounded until an adversarially chosen time GST and are bounded thereafter. In this model, the protocol is secure only after $O(\mathsf{GST})$ time [27]. In the sleepy model, the adversary has the power to set honest parties to sleep for arbitrarily long; these sleepy parties suffer unbounded delay. The longest-chain protocol is secure in the sleepy model, provided the fraction of awake honest parties always exceeds the fraction of corrupt parties [23].

A key point in the above models is that for those parties suffering from unbounded delay, it is impossible to provide any security guarantees. In comparison, for the random delay model, we are able to provide security guarantees that hold for any (finite set of) parties at all times. This is possible precisely because delays are random; it is unlikely that a given party suffers large delays for a long period of time. In conclusion, each of these models capture different facets of sub-optimal

network behavior. For a real-world network, the appropriate model may be any one of these, and may even change over time. Moreover, these models are not exclusive; e.g., in the sleepy model, the awake parties can be modeled as having random delays.

Two other works [28, 29] study the longest-chain protocol under random, unbounded delay. These works model the network as a graph of inter-connected nodes, and assume that delays between two neighboring nodes in the network are exponentially distributed and i.i.d. In this work, we only model point-to-point communication (equivalently, we assume the network is a complete graph). However, we allow for arbitrary delay distributions. More importantly, these chapters do not consider the security of the protocol against an adversary. They are thus different in scope.

Security Statements Many chapters on the security analysis of the longest-chain protocol consider both a safety property and a liveness property. However, there are minor differences in the precise statements used. The settlement (safety) property is inspired from that of Blum et al. [25]. Moreover, following Blum et al. [25], we state this property in both intensive and extensive forms, and show that guarantees for the former lead to guarantees for the latter. The chain quality (liveness) property, in its extensive form, is the same as the existential chain quality property of Badertscher et al. [22]. Compared to them, we obtain tighter bounds for the same property, because of our proof technique of obtaining bounds for intensive properties and then deriving bounds for the extensive ones.

Security Analysis At a high level, our security analysis follows that of other works analyzing the PoW longest-chain protocol [4, 6, 30, 8, 9, 10]. I.e., we first use deterministic, combinatorial arguments to map events concerning chains to events concerning some random processes, and then use probabilistic analysis to bound their atypical behavior. The novelty of this chapter lies in constructing random objects that, on the one hand, give sufficient conditions for security in the random delay model, and on the other hand, are amenable to probabilistic analysis.

The security threshold obtained in this work is not tight, as can be seen by comparing our security threshold with (2.1) for the special case of a constant Δ. The works of Dembo et al. [7] and Gaži et al. [8] give a tight characterization of the security regime in the synchronous model. Extending the results of [7, 8] to the random delay model is a direction of future research.

2.2 The System Model

This section contains the mathematical model for a PoW blockchain system. This includes the protocol specifications (Section 2.2.2), adversarial powers (Section 2.2.2), the leader election mechanism (Section 2.2.3) and the communication network model (Section 2.2.4). The novel aspects of the model are the random network delays and the one-time leader election model, described in Sections 2.2.4 and 2.2.3 respectively. The other elements of our model are similar to those studied in earlier work.

2.2.1 Preliminaries

The longest-chain protocol is run by a set of parties that aim to achieve consensus (or agreement) on an ever-growing sequence of *blocks*. The protocol proceeds in discrete time slots that are indexed by $\mathbb{N}$ and runs for an infinite duration. We assume that clocks of all parties are perfectly synchronized. Blocks are treated as abstract data structures containing an integer timestamp, a hash pointer to a parent block with a smaller timestamp, a cryptographic signature of the block's proposer, some transactions and other relevant information. A special genesis block, with timestamp zero and no parent, is known to all parties at the start of the protocol.

Parties in the protocol The parties in the protocol comprise of multiple honest ones and a single adversary $\mathcal{A}$. (Replacing all corrupt parties by a single one is done for simplicity). The set of honest parties is represented by $\mathcal{H}$ and may be finite or infinite. Arbitrary honest parties are denoted by h, h_1, h_2, etc. In our model, the adversary can never corrupt an honest party and the honest parties never go offline. Honest parties follow the longest-chain protocol, while the adversary can deviate from the protocol in certain ways. The prescribed behavior for honest parties and the powers of the adversary are described in Section 2.2.2.

Blockchains From any block, a unique sequence of blocks leading back to the genesis block can be identified via the hash pointers. We call this sequence a *blockchain*, or simply a *chain*. A generic chain is denoted by $\mathcal{C}$. The convention is that the genesis block is the first block of the chain, and the terminating block is called the *tip*. The timestamps of blocks in a blockchain must strictly increase, going from the genesis to the tip. At any given slot, honest parties store a single chain in their memory. We use $\mathcal{C}_i^h$ to denote the chain held by an honest party h at (the end of) slot i. We use $\mathcal{C}[i_1 : i_2]$ to represent the portion of a chain $\mathcal{C}$ consisting of blocks with timestamps in the interval $\{i_1, \ldots, i_2\}$. The length of a chain $\mathcal{C}$, denoted by $|\mathcal{C}|$, is defined to be the number of blocks in the chain, excluding the genesis block. Likewise, $|\mathcal{C}[i_1 : i_2]|$ denotes the number of blocks in $\mathcal{C}$ that are mined in the interval $\{i_1, \ldots, i_2\}$.

Blocktrees The set of all blocks generated up to a given slot i forms a directed tree. Let $\mathcal{F}_i$ be the directed graph (V, E), where V is the set of blocks generated up to slot i and E is the set of parent-child block pairs. These edges point from parent to child, in the opposite direction of the hash pointers. The genesis block is the root of the tree, with no parent. In addition, the timestamp of block v is denoted by $\ell(v)$. Every blockchain $\mathcal{C}_i^h$ is a directed path in $\mathcal{F}_i$ that begins at the genesis block and ends at any other block. $\mathcal{F}_i$ also includes blocks held privately by the adversary. The adversary stores the entire blocktree $\mathcal{F}_i$ in its memory.

2.2.2 Details of a Slot

Within a slot, the following events occur in the given order. This describes the prescribed protocol for honest parties. It also specifies the adversary's powers.

- (**Leader Election Phase**) The leader election mechanism chooses at most one party, either honest or adversarial, as the leader of the slot. The choice of the leader is made at random, independently of the past. All parties know whether or not they are the leader of the slot.
- (**Honest Send Phase**) If an honest party is chosen as a leader, it creates a new block with the timestamp of the current slot, appends it to its chain, and broadcasts this new chain to all parties. The communication network assigns an independent random delay for this message's delivery to each honest recipient.
- (**Adversarial Send Phase**) $\mathcal{A}$ receives the chain sent (if any) in the Honest Send Phase, and also learns of the delays assigned by the network for each recipient. If $\mathcal{A}$ is the leader of the slot, it creates a single new block, extending any chain in $\mathcal{F}_i$. It may then send the new block (along with the preceding blockchain) to an arbitrary subset of honest parties, or hold it privately. It can also choose to send blocks it mined in the past, which it has held privately, to some honest parties.
- (**Deliver Phase**) Messages from honest parties slated for delivery in the current slot and $\mathcal{A}$'s messages from the current slot are delivered to the appropriate honest parties. $\mathcal{A}$ can also choose to deliver any honest messages ahead of schedule.
- (**Adopt Phase**) Each honest party updates its chain if it receives any chain strictly longer than the one it holds. It chooses the longest one among the ones it receives, with $\mathcal{A}$ breaking any ties. These actions give the protocol its name.

2.2.3 Leader Election Mechanism

The leader election mechanism is an idealized model of the PoW mining process (see Garay et al. [4] for more details about mining). In each slot, the mechanism chooses at most one leader among the parties at random, whose role is to propose a new block. This random process can be described as follows. Whether or not a slot is empty forms a Bernoulli process, i.e., a sequence of independent and identically distributed (i.i.d.) Bernoulli random variables. Let f be the probability of a nonempty slot. In particular, this implies that the interval between two nonempty slots is geometrically distributed with parameter f (which we denote by $\mathsf{geom}(f)$). In PoW parlance, f is the mining rate of the protocol. Further, among the nonempty slots, the sequence of honest and adversarial leaders also forms a Bernoulli process. Given that a slot is nonempty, let α be the probability that the leader is honest; the leader is $\mathcal{A}$ with probability $\beta = 1 - \alpha$. Thus, α is akin to the fraction of

honest mining power in the system. Such a leader election process is also considered by Gaži et al. [8].

Within this framework, we consider two slightly different leader election models, namely the *i.i.d. leader election model* and the *one-time leader election model*. The difference is in terms of the identities of the honest leaders. In the former, the sequence of honest leaders is i.i.d. In particular, it is possible that the same honest party is elected twice in a row. In the latter, each honest party can be a leader at most once. In particular, this implies that there are infinitely many honest parties. The model is the asymptotic limit of a system with a large number of miners with similar mining powers.

Arguably, the i.i.d. leader model is more realistic than the one-time leader model. However, the security analysis of the former has some subtleties. These arise from the possibility that the same honest party is elected a leader multiple times in close succession. Therefore, we prove our results for the one-time leader model in the main text, and mention the necessary additional arguments for the i.i.d. leader model in the appendix. Interestingly, the model with random delays forces us to pay attention to the identities of the honest leaders. In contrast, these identities do not appear in the analysis in the synchronous model [25, 8]. Finally, note that we only consider the possibility of zero or one leader per slot; this is for simplicity's sake. Strictly speaking, a discretized version of the continuous-time mining process should allow for multiple leaders to be elected in a slot, but this event can be made negligibly rare by taking the slot duration to be small enough.

2.2.4 The Communication Model

The communication network is modeled as follows. Every message sent by one honest party to another is subject to a random delay, which can take any value in $\mathbb{Z}_+$. We consider a broadcast to be a set of different point-to-point messages, each of which is subject to an independent delay. We adopt the convention that the minimum possible delay is zero; in this case, a message sent in a time slot is received by the end of that slot. The delays of different messages are i.i.d. Let Δ denote a random variable with the common distribution, which we call the *delay distribution*. The synchronous model is a special case with a constant Δ.

In the i.i.d. leader election model, we require that the delay distribution has a *nondecreasing failure rate function*. The failure rate function for the delay distribution Δ is defined as

$$\mathsf{FailureRate}[s] = \begin{cases} \mathbb{P}(\Delta = s | \Delta \geq s) & \text{if } \mathbb{P}(\Delta \geq s) > 0 \\ 1 & \text{if } \mathbb{P}(\Delta \geq s) = 0 \end{cases}$$

Roughly speaking, the requirement implies that the longer one has waited for a message, the sooner it is likely to arrive. A geometric random variable has a constant failure rate. A constant Δ has a failure rate function that is zero up to the constant and one thereafter. Therefore, they are both admissible. However, the condition precludes heavy-tailed distributions. This technical condition

is required to handle the subtlety mentioned in Section 2.2.3 above–the case where the same leader is elected multiple times in close succession. Note that the one-time leader model does not require these conditions. It even permits distributions with positive probability for $\Delta = \infty$, i.e., where messages are not delivered.

Given that messages in a blockchain network are spread via gossip protocols, the point-to-point delay distribution is likely to have a geometric tail. Thus, we argue that the non-negative failure rate assumption is not too unrealistic. Moreover, given the power of the adversary to deliver honest messages earlier than scheduled, a system with a given delay distribution Δ can be emulated by a system that has a different delay distribution $\tilde{\Delta}$, provided the latter stochastically dominates the former. If $\tilde{\Delta}$ satisfies the nondecreasing failure rate restriction, guarantees for a system with delay Δ can be given in terms of the distribution $\tilde{\Delta}$.

2.3 The Desired Security Properties

This section contains the formal definitions of our security properties, the condition under which the properties hold (with high probability), and the main theorem which bounds the probability that these are violated. The two security properties are the safety property (settlement/common-prefix) and the liveness property (chain quality). Both are stated in intensive and extensive forms, and these forms are compared. The security properties and the guarantees for them hold for both models introduced in Section 2.2: the i.i.d. leader model with delays having a non-decreasing failure rate function and the one-time leader model with general delay distributions.

Each security property refers to a desirable condition over an *execution*. An execution of the protocol refers to a particular instantiation of the random components (i.e., leader election and communication delays) and the actions of the adversary. Whether a certain property holds or not in an execution depends on both these factors. The adversary's actions can be arbitrary and our theorems are stated for the worst-case scenario of all possible adversarial actions.

2.3.1 Property Definitions

We first define the settlement property and then the common prefix property. These are intensive and extensive forms of safety respectively, and are borrowed from Blum et al. [25].

Definition 2.1 (Settlement)**.** *In an execution, the* settlement property with parameters $s, k \in \mathbb{N}$ and $\mathcal{I} \subseteq \mathcal{H}$ *holds if, for any pair of honest parties* $h, h' \in \mathcal{I}$ *and slots* i, i' *such that* $s + k \leq i \leq i'$*, it holds that* $\mathcal{C}_i^h[1:s] = \mathcal{C}_{i'}^{h'}[1:s]$.

We refer to the settlement property with parameters s, k, and $\mathcal{I}$ as the $(s, k, \mathcal{I})$-settlement property for brevity. A similar convention is used for other properties too. The $(s, k, \mathcal{I})$-settlement property states that parties in $\mathcal{I}$ will agree on the order of blocks mined up to slot s after k more slots.

Definition 2.2 (Common Prefix)**.** *In an execution, the* common prefix property with parameters $T, k \in \mathbb{N}$ and $\mathcal{I} \subseteq \mathcal{H}$ *holds if, for any pair of honest parties* $h, h' \in \mathcal{I}$ *and slots* s, i, i' *such that* $s \leq T$ *and* $s + k \leq i \leq i'$*, it holds that* $\mathcal{C}_i^h[1:s] = \mathcal{C}_{i'}^{h'}[1:s]$.

The intensive and extensive forms of safety have a subtle difference, which is illustrated with the following example. Let T be some large number. The $(T, k, \mathcal{I})$-settlement property means the parties in $\mathcal{I}$ agree forever after slot $T + k$ about the chain up to slot T. This immediately implies that all parties in $\mathcal{I}$ agree forever about the chain up to slot s, after slot $T + k$, for any $s \leq T$. This does not, however, imply that all parties in $\mathcal{I}$ agree forever about the chain up to time s, after slot $s + k$, for all s with $s \leq T$. This latter statement, which is stronger, is captured by the extensive form given by the common prefix property. The following lemma relates the two forms of safety.

Lemma 2.1. *Fix a set of honest parties* $\mathcal{I} \subseteq \mathcal{H}$ *and parameters* $T, k \in \mathbb{N}$*. The* $(T, k, \mathcal{I})$*-common prefix property holds if and only if the* $(s, k, \mathcal{I})$*-settlement property holds for all* $s \leq T$*.*

Proof. Pick any pair of honest parties, $h, h' \in \mathcal{I}$ and any slot $s \leq T$. Pick any i, i' satisfying $s + k \leq i \leq i'$. Consider the chains held by h, h' at slots i, i' respectively: $\mathcal{C}_i^h, \mathcal{C}_{i'}^{h'}$. The proof follows from the definitions of the common prefix and settlement properties, and the following equivalences: $(T, k, \mathcal{I})$-common prefix property holds $\Leftrightarrow \ \forall\, s \leq T, \mathcal{C}_i^h[1:s] = \mathcal{C}_{i'}^{h'}[1:s] \Leftrightarrow \ \forall\, s \leq T$, $(s, k, \mathcal{I})$-settlement property holds. □

We next state the chain quality property, first in its intensive form and then in its extensive form. In some works, e.g., [22], (the extensive form of) this property has been called the existential chain quality property, while chain quality refers to a more general form of the property. We present the simpler form in this chapter.

Definition 2.3 (Intensive Chain Quality)**.** *In an execution, the* intensive chain quality property *with parameters* $s, k \in \mathbb{N}$ *and* $\mathcal{I} \subseteq \mathcal{H}$ *holds if, for any honest player* $h \in \mathcal{I}$ *and slot* $i \geq s + k$*,* $\mathcal{C}_i^h[s+1:s+k]$ *contains at least one honestly mined block.*

Definition 2.4 (Extensive Chain Quality)**.** *In an execution, the* extensive chain quality property *with parameters* $T, k \in \mathbb{N}$ *and* $\mathcal{I} \subseteq \mathcal{H}$ *holds if, for any honest player* $h \in \mathcal{I}$ *and slots* s, i *such that* $s \leq T$ *and* $i \geq s + k$*,* $\mathcal{C}_i^h[s+1:s+k]$ *contains at least one honestly mined block.*

The relation between the intensive and extensive versions of chain quality parallels that between the settlement and common prefix property noted in Lemma 2.1, and is stated next (without proof).

Lemma 2.2. *Fix a set of honest parties* $\mathcal{I} \subseteq \mathcal{H}$ *and parameters* $T, k \in \mathbb{N}$*. The* $(T, k, \mathcal{I})$*-extensive chain quality property holds if and only if the* $(s, k, \mathcal{I})$*-intensive chain quality property holds for all* $s \leq T$*.*

Note that the subset of honest parties $\mathcal{I}$ is explicitly mentioned in the security properties. This is because the probability with which the properties are violated increases with the size of this set (see

Theorem 2.1). This is inevitable in the random delay model with unbounded delay distributions. Consider a setting with infinitely many honest parties in a system with geometrically distributed delays. Then, for any fixed T, there will almost surely exist an honest party that has not heard of any honest message up to slot T (this can be shown by the second Borel-Cantelli Lemma). Clearly, such a party cannot be guaranteed to have any security results. This issue does not arise if delays are bounded. Therefore, in earlier works that deal with the constant Δ setting [4, 20], the subset of honest parties $\mathcal{I}$ is implicitly taken to be the entire set $\mathcal{H}$.

2.3.2 Main Result

The main result of this chapter, Theorem 2.1, is a bound on the probability that either of the intensive security properties are violated. Naturally, such a security result can be proven only under the condition that the fraction of honest mining power is large enough (at least a half), and that the typical delay is small enough compared to the mining rate. In this work, the security results hold under the *ϵ-honest majority* condition, which is defined next.

Definition 2.5 (ϵ-honest majority)**.** *Consider a blockchain protocol where the leader election process has parameters α and f; and the network's delay is represented by a random variable Δ. Let $G \sim \mathsf{geom}(f)$ be a random variable that is independent of Δ, which denotes the typical interval between nonempty slots. Let $p \triangleq \alpha\,\mathbb{P}(\Delta < G)$. Suppose the system's parameters are such that $p > 0.5$. Let ϵ be such that $p = (1+\epsilon)/2$. We say that such a protocol has* ϵ-honest majority*.*

Theorem 2.1 (Main Result)**.** *Consider a blockchain protocol with ϵ-honest majority. Then for any $\mathcal{I} \subseteq \mathcal{H}$, $s \in \mathbb{N}$ and $k \in \mathbb{N}$,*

$$\mathbb{P}((s,k,\mathcal{I})\text{-settlement property is violated OR } (s,k,\mathcal{I})\text{-intensive chain quality property is violated}) \leq p_{\mathsf{char-string}} + |\mathcal{I}| p_{\mathsf{unheard}} \quad (2.2)$$

where

$$p_{\mathsf{char-string}} = 4\exp(-(3/32)\epsilon^2 fk) \qquad p_{\mathsf{unheard}} = (8/\epsilon)\exp(-(1/32)\epsilon fk).$$

As a corollary, we get the following result about the extensive safety and liveness properties.

Corollary 2.1. *Consider a blockchain protocol with ϵ-honest majority. Then for any $\mathcal{I} \subseteq \mathcal{H}$, $T \in \mathbb{N}$ and $k \in \mathbb{N}$,*

$$\mathbb{P}((T,k,\mathcal{I})\text{-common prefix property is violated OR } (T,k,\mathcal{I})\text{-extensive chain quality property is violated)}) \leq T\,(p_{\mathsf{char-string}} + |\mathcal{I}| p_{\mathsf{unheard}})$$

The key difference between intensive and extensive properties can be seen by the guarantees on them. The probability of the intensive properties being violated is bounded independently of

T (Theorem 2.1). The bound on the probability of an extensive properties being violated grows linearly with T (Corollary 2.1). Corollary 2.1 follows from Theorem 2.1, Lemmas 2.1 and 2.2, and the union bound. We omit a formal proof.

Theorem 2.1 and its corollary prove security properties under the ϵ-honest majority condition for some $\epsilon > 0$. By setting $\epsilon = 0$, we get a lower bound on the security threshold of a system parameterized by (f, Δ). (We get a lower bound since the ϵ-honest majority condition is sufficient, but not necessary, for security.) Recall that the security threshold is the supremum of the adversarial power that can be tolerated as the function of the system's mining rate and delay parameters. Note that for any random Δ such that $\mathbb{P}(\Delta < \infty) = 1$, one can choose f small enough such that $\mathbb{P}(\Delta < G)$ is very close to one, which means that the security threshold can be made arbitrarily close to one-half (which is optimal).

While the results are stated for discrete time, they imply corresponding results in continuous time by taking a limit, as described in [8]. Also, in continuous time, the probability of more than one party mining at a time is zero. Leaders are elected at times of a Poisson process of rate f, with a leader being the adversary with probability β and honest with probability $1 - \beta$. The honest majority condition reduces to $(1 - \beta)\mathbb{P}(\Delta < \mathsf{Exp}(f)) > \frac{1}{2}$, where $\mathsf{Exp}(f)$ denotes an exponentially distributed random variable with rate parameter f (mean $1/f$). In case Δ has the $\mathsf{Exp}(1/\eta)$ distribution (with mean η), the honest majority condition becomes $\beta < \frac{1-\eta f}{2}$.

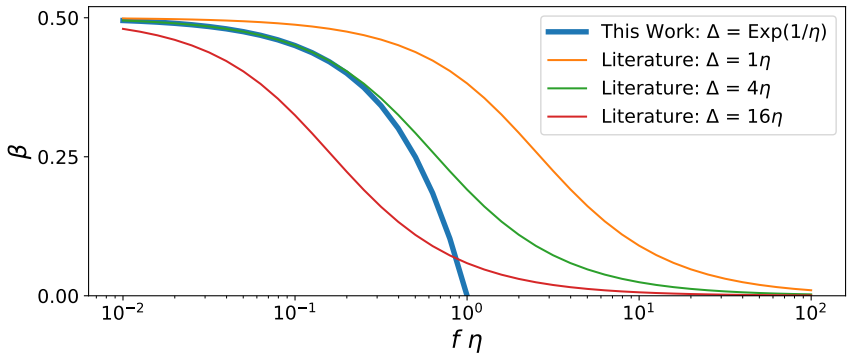

Figure 2.1: Comparison of the security threshold guaranteed by this work for exponentially distributed delays with the tight threshold for deterministic delays (i.e., (2.1)).

Figure 2.1 displays the security threshold from this work for delays being exponentially distributed random variables $\Delta \sim \mathsf{Exp}(1/\eta)$, and for comparison, the boundaries of the security region guaranteed for bounded delay by (2.1) for $\Delta \equiv \eta$, $\Delta \equiv 4\eta$, and $\Delta \equiv 16\eta$. The figure shows that the adversarial tolerance guarantees provided by this work with $\mathsf{Exp}(1/\eta)$ delays are better than the best possible guarantees with constant delays 16η in the range $f\eta < 0.8$. Consider, for the sake of illustration, a system whose delays are random with a distribution that is identical to $\mathsf{Exp}(1/\eta)$ from $[0, 16\eta]$, and concentrates the rest of the mass at 16η. Such a distribution is stochastically dominated by both $\mathsf{Exp}(1/\eta)$ and the constant delay 16η, and, thus, guarantees for both models apply. Thus, for such a system, the security region of the protocol as guaranteed by Theorem 2.1

is greater than what can be established via the synchronous model alone.

2.3.3 Proof Sketch

The basic intuition guiding the security analysis in this work is the following. The security properties are likely to be violate if the adversary can build two equally long chains that fork far back in time. There are two factors that contribute to long forked chains, namely, adversarial blocks and network delays amongst honest parties. To elaborate, the adversary can purposely deviate from the longest-chain rule and use its blocks to extend the shorter of two competing chains. In addition, due to network delays, it is possible that some honest parties extend the shorter of two forked chains, simply because they have not heard of the latest block(s) on the longer chain. If the number of adversarial blocks and the number of forked (or parallel) honest blocks are small relative to the length of the blockchain, the protocol remains secure.

To distinguish between honest blocks that grow the chain from those that do not, we introduce the notion of *special honest slots*. These are recursively defined as follows. Given a special honest slot s, the first honest slot after s whose leader hears of the block mined at s is termed special honest. It immediately follows that blocks from special honest slots are always built at ever-increasing heights on the blockchain, contributing to the blockchain's growth. All other honest blocks are treated as adversarial, as they could be placed arbitrarily in the blocktree.

Whether or not a given honest slot s is special honest is determined by two random entities: the time passed since the last special honest slot, and the delay in conveying that block to the leader of s. Both these factors are beyond the adversary's control; they depend solely on the mining rate f and the delay distribution Δ. The ϵ-honest majority condition required for security (Definition 2.5) is simply the requirement that every new block is special honest with probability at least a half. If this condition holds, then over any considerable interval, the number of special honest slots exceed the number of adversarial slots with high probability. We show that this is sufficient to guarantee security.

The rest of the chapter is devoted to the proof of Theorem 2.1. Section 2.4 introduces some key notation and the definition of special honest slots. The proof is split into two main components. First, via a deterministic, combinatorial analysis, we obtain a sufficient condition for security, $\mathcal{E}_{\mathsf{secure}}$ (Section 2.5). Second, we provide an upper bound on the probability of $\mathcal{E}^c_{\mathsf{secure}}$ occurring, which in turn provides the desired security guarantee (Sections 2.6 and 2.7).

2.4 Definitions of Key Random Processes

All random processes in our model are discrete-time processes, indexed by $\mathbb{N}$, $\mathbb{Z}_+$ or $\mathbb{Z}$ (the relevant indexing will be specified when the process is defined). For a random process $\mathsf{Process}$, the notation for the i^{th} variable is $\mathsf{Process}[i]$. Denote an interval of slots $\{i_1, \ldots, i_2\}$ by $[i_1 : i_2]$. The portion of

the process over such an interval is denoted by $\mathsf{Process}[i_1 : i_2]$. If $i_2 < i_1$, $[i_1 : i_2]$ (and by extension, $\mathsf{Process}[i_1 : i_2]$) denotes an empty interval (or string).

2.4.1 LeaderString as a stationary renewal process

We start by defining LeaderString, a random process which represents the sequence of leaders in each slot without the leader identities. We then extend this process to negative time and note that it has a stationary renewal structure. LeaderString takes values in $\{\perp, 0, 1\}$. For $i \geq 1$,

$$\mathsf{LeaderString}[i] = \begin{cases} \perp & \text{if slot } i \text{ has no leader} \\ 0 & \text{if slot } i \text{ has an honest leader} \\ 1 & \text{if slot } i \text{ has the adversary as leader} \end{cases} \tag{2.3}$$

LeaderString is an i.i.d. process with $\mathbb{P}(\mathsf{LeaderString}[i] = \perp) = 1 - f$, $\mathbb{P}(\mathsf{LeaderString}[i] = 0) = \alpha f$ and $\mathbb{P}(\mathsf{LeaderString}[i] = 1) = (1 - \alpha)f$ (see Section 2.2.3). Extend this process to negative time by defining $(\mathsf{LeaderString}[i] : i \leq 0)$ to be a sequence of i.i.d. random variables with the same probabilities as noted above. Call all slots with $\mathsf{LeaderString}[i] \neq \perp$ as *nonempty* slots and those with $\mathsf{LeaderString}[i] = 0$ as *honest slots*. Let $1 \leq T_1 < T_2 < \ldots$ denote the indices of the nonempty slots of LeaderString from slot 1 onward. Similarly, let $0 \geq T_0 > T_{-1} > T_{-2} > \ldots$ index the nonempty slots before or up to slot zero, going backwards in time. For any interval $[i_1 : i_2]$, define $N[i_1 : i_2]$ to be the number of nonempty slots in that interval.

With the above extension, LeaderString is endowed with a stationary renewal structure. A *stationary process* is one whose distribution is invariant under any time shift. A *renewal process* is an arrival process (a sequence of events or arrivals) where the interval between two arrivals are i.i.d. The times of the arrivals are called renewal points, and the gap between renewal points is called the lifetime. Such a process is characterized entirely by the lifetime distribution.

The set of nonempty slots forms a *stationary renewal process* with lifetime distribution $\mathsf{geom}(f)$. Given the locations of all the renewal points, the labels at the renewal points are i.i.d. Bernoulli random variables with $\mathbb{P}(0) = \alpha$. For any $j \neq 1$, $T_j - T_{j-1}$ has distribution $\mathsf{geom}(f)$, while $T_1 - T_0 = T_1 + (1 - T_0) - 1$, so that $T_1 - T_0$ is the sum of two independent $\mathsf{geom}(f)$ random variables minus one. In the terminology of renewal theory, $T_1 - T_0$ is the *sampled lifetime* sampled at time 0. In the Ouroboros line of works [20, 21, 22], the process LeaderString is called the characteristic string. We reserve that term for a slightly different process, which is defined next.

2.4.2 Special Honest Slots and CharString

We now introduce a new concept called *special honest slots*, which can be thought of as a generalization of Δ-isolated slots from [21]. Special honest slots are a subset of slots with honest leaders, defined in a recursive manner, such that the leader of each special honest slot must have heard of

the block broadcast in the previous special honest slot. Simultaneously, we also define the notion of *the characteristic string*, denoted by $\mathsf{CharString}$. The process $\mathsf{CharString}$ is a process indexed by $\mathbb{Z}$, taking values in $\{\perp, 0, 1\}$. In $\mathsf{CharString}$, special honest slots are marked with symbol 0, other non-empty slots with symbol 1, and empty slots with the symbol $\perp$. Thus, $\mathsf{CharString}$ is obtained from $\mathsf{LeaderString}$, by converting a few 0 symbols to 1 (those honest slots that are not special).

The precise definition of $\mathsf{CharString}$ for the one-time leader model is given in the remainder of this subsection. (The construction in the i.i.d. leader model is given in Appendix 2.8). We introduce some notation for convenience. For all $j > 0$ where T_j is an honest slot, let h_j denote the leader of the slot. Let $\mathsf{delay}(T_j \to h)$ denote the delay in the message sent by h_j in slot T_j to an honest party $h \in \mathcal{H}$. Finally, let $\mathsf{delay}(T_j \to h)$ be dummy random variables with the same distribution as Δ for all honest slots $T_j < 0$. In the construction of $\mathsf{CharString}$, consider the entire process $\mathsf{LeaderString}$ (in particular, $\{T_j : j \in \mathbb{Z}\}$) and all the above defined quantities as given.

For all $i \in \mathbb{Z}$ such that $\mathsf{LeaderString}[i] \in \{1, \perp\}$, let $\mathsf{CharString}[i] = \mathsf{LeaderString}[i]$. For $j \leq 0$, if $\mathsf{LeaderString}[T_j] = 0$, let $\mathsf{CharString}[T_j]$ be 0 with probability $\mathbb{P}(\Delta < T_j - T_{j-1} | T_j - T_{j-1})$, and let $\mathsf{CharString}[T_j]$ be 1 otherwise. The aforementioned choices are conditionally independent across all $j \leq 0$. For $j \geq 1$, let T_{j^*} denote the last special honest slot strictly before T_j. Let $\mathsf{CharString}[T_j] = 0$ if slot T_j is honest and $R_j < T_j - T_{j-1}$, where $R_j \triangleq \mathsf{delay}(T_{j^*} \to h_j)$. Note that the $\mathsf{CharString}$ process for negative time was used to define the first special honest slot in positive time. This concludes the construction of $\mathsf{CharString}$, given $\mathsf{LeaderString}$, the identities of the honest leaders h_j, and the delays amongst all honest parties $\mathsf{delay}(T_j \to h)$. Later, in Section 2.6.2, we characterize the distribution of $\mathsf{CharString}$. The definition given here endows it with a stationary renewal structure, which is useful in our security analysis.

Special honest slots are the slots T_j where $\mathsf{CharString}[T_j] = 0$. Those nonempty slots that are not special honest are defined to be adversarial. Note that this includes those slots that are honest, but not special honest. For any interval $[i_1 : i_2]$, define $N_{\mathsf{spl}}[i_1 : i_2]$ to be the number of special honest slots, and let $N_{\mathsf{adv}}[i_1 : i_2]$ denote the number of adversarial slots, in that interval. It follows that $\mathsf{N}[i_1 : i_2] = N_{\mathsf{spl}}[i_1 : i_2] + N_{\mathsf{adv}}[i_1 : i_2]$. Special honest slots have the following key property: blocks mined in successive special honest slots are built at strictly increasing heights. Note that h_j receives the message from the previous special honest slot at time $T_{j^*} + R_j$. If T_j is special honest, then$T_{j^*} + R_j < T_{j^*} + T_j - T_{j-1} \leq T_j$. Thus, the condition for T_j to be a special honest slot is sufficient, but not necessary, for h_j to have received the message from the previous special honest slot. From this, the aforementioned property follows. This property is used to lower bound the length of the portion of chains in Lemmas 2.3 and 2.4. It is also used in Lemma 2.5 to derive a necessary condition for settlement violation.

2.4.3 The Unheard process

For an honest party h and $i \geq 1$, let $\mathsf{LatestHeard}_h[i]$ denote the special honest slot with greatest index whose broadcast h has heard by the end of slot i. That is,

$$\mathsf{LatestHeard}_h[i] = \max\{T_j : 1 \leq T_j \leq i, \mathsf{CharString}[T_j] = 0, T_j + \mathsf{delay}(T_j \to h) \leq i\}. \tag{2.4}$$

We use the convention that the maximum of an empty set is zero. Thus, $0 \leq \mathsf{LatestHeard}_h[i] \leq i$.

Let $\mathsf{Unheard}_h[i]$ denote the number of consecutive special honest slots that party h has not heard of by slot i, counting backwards from i. That is, $\mathsf{Unheard}_h[i]$ is the number of special honest slots in the interval $[\mathsf{LatestHeard}_h[i] + 1, i]$. Mathematically,

$$\mathsf{Unheard}_h[i] = N_{\mathsf{spl}}[\mathsf{LatestHeard}_h[i] + 1 : i]. \tag{2.5}$$

For example, $\mathsf{Unheard}_h[i] = 2$ means that by the end of slot i, h had not heard the last two special honest slots occurring before or at i; it either heard the third most recent special honest slot before slot i or there were only two special honest slots during $[1 : i]$. Looking ahead, $\mathsf{Unheard}_h[i]$ is used to lower bound the length of the chain $\mathcal{C}_i^h$, as shown in Lemma 2.4. In turn, this lower bound is used to derive necessary conditions for settlement and chain quality violation in Lemmas 2.5 and 2.6.

Finally, given a set of honest parties $\mathcal{I}$, let

$$\mathsf{LatestHeard}_{\mathcal{I}}[i] = \min_{h \in \mathcal{I}} \mathsf{LatestHeard}_h[i], \quad \mathsf{Unheard}_{\mathcal{I}}[i] = \max_{h \in \mathcal{I}} \mathsf{Unheard}_h[i].$$

I.e., $\mathsf{Unheard}_{\mathcal{I}}[i]$ is an upper bound for $\mathsf{Unheard}_h[i]$ for all $h \in \mathcal{I}$.

2.5 Lemmas on Deterministic Properties

This section gives some necessary conditions for the settlement property and the chain quality property to be violated. The conditions are expressed in terms of the CharString and Unheard processes: they must hold if the adversary has violated the security properties. The necessary conditions for both the security properties turn out to be identical. It can be worded as follows: over some sizable interval, the number of non-special honest blocks must exceed the number of special honest blocks, barring a term of Unheard (see (2.7) for the exact condition). This analysis factors out the arbitrary actions of the adversary, or equivalently, considers the worst-case adversarial actions. With this analysis, the proof of Theorem 2.1 is reduced to bounding the probability of some event concerning the random processes CharString and Unheard.

Before formally proving these reductions, we introduce the notion of the *latest special honest chain* (at slot i), and use this notion to give two forms of the *chain growth* property. Such a property has been used by other works that analyze PoW blockchains [4, 6] in their security analysis.

Definition 2.6 (Latest Special Honest Chain)**.** *For any slot $i \in \mathbb{N}$, let $\mathcal{C}_i^*$ denote the chain broadcast by the leader of the last special honest slot at or before slot i. Call this the latest special honest chain at slot i. In $\mathcal{F}_i$, $\mathcal{C}_i^*$ is the longest chain that ends in a special honest block. $\mathcal{C}_0^*$ is the genesis block.*

Two simple facts follow immediately from the definition. First, if i is a special honest slot, then $\mathcal{C}_i^*$ is the chain that is broadcast by the leader of slot i. Second, if a chain $\mathcal{C}$ contains a block mined in a special honest slot s, then $\mathcal{C}[1:s] = \mathcal{C}_s^*$. These facts are used frequently in the following proofs. The following chain growth lemma is also easily proven.

Lemma 2.3 (Chain Growth–Special Honest)**.**

$$|\mathcal{C}_{i'}^*| \geq |\mathcal{C}_i^*| + N_{\mathsf{spl}}[i+1:i'], \quad \forall\ i \leq i'. \tag{2.6}$$

Proof. The property of special honest slots implies that the chains broadcast in successive special honest slots must have strictly increasing lengths. This implies (2.6). □

As it stands, the above lemma only lower bounds the growth of latest special honest chains, $\mathcal{C}_i^*$. However, it is useful in bounding the chain growth of any honestly held chain, as shown next.

Lemma 2.4 (Chain Growth–Honest)**.** *For any $i \in \mathbb{N}$ and any $h \in \mathcal{H}$, suppose $\mathcal{C}_i^h$ contains a special honest block mined in slot r, or let $r = 0$. Then*

$$|\mathcal{C}_i^h| \geq |\mathcal{C}_r^*| + N_{\mathsf{spl}}[r+1:i] - \mathsf{Unheard}_h[i].$$

Equivalently,

$$|\mathcal{C}_i^h[r+1:i]| \geq N_{\mathsf{spl}}[r+1:i] - \mathsf{Unheard}_h[i].$$

Proof. By the definition of $\mathsf{LatestHeard}$ (see Section 2.4.3), the honest party h must have heard of the chain $\mathcal{C}_l^*$ by slot i, where $l \triangleq \mathsf{LatestHeard}_h[i]$. Therefore, the chain it holds at slot i must be at least as long as $\mathcal{C}_l^*$, i.e., $|\mathcal{C}_i^h| \geq |\mathcal{C}_l^*|$. By Lemma 2.3, $|\mathcal{C}_l^*| \geq |\mathcal{C}_r^*| + N_{\mathsf{spl}}[r+1:l]$, provided $l \geq r$. We know that $\mathsf{Unheard}_h[i] = N_{\mathsf{spl}}[l+1:i]$ (by (2.5)). Putting the above inequalities together, we get

$$\begin{aligned} |\mathcal{C}_i^h| \geq |\mathcal{C}_l^*| &\geq |\mathcal{C}_r^*| + N_{\mathsf{spl}}[r+1:l] \\ &= |\mathcal{C}_r^*| + N_{\mathsf{spl}}[r+1:i] - N_{\mathsf{spl}}[l+1:i] \\ &= |\mathcal{C}_r^*| + N_{\mathsf{spl}}[r+1:i] - \mathsf{Unheard}_h[i]. \end{aligned}$$

This proves the first statement, under the condition $l \geq r$. If $l < r$,

$$N_{\mathsf{spl}}[r+1:i] \leq N_{\mathsf{spl}}[l+1:i] = \mathsf{Unheard}_h[i].$$

Under this condition, the lemma is trivially true: $\mathcal{C}_i^h$ must be at least as long as $\mathcal{C}_r^*$, as it contains the block mined in slot r.

The second statement can be obtained from the first, as follows. Observe that $\mathcal{C}_i^h[1:r] = \mathcal{C}_r^*$, and therefore

$$|\mathcal{C}_i^h| = |\mathcal{C}_i^h[1:r]| + |\mathcal{C}_i^h[r+1:i]| = |\mathcal{C}_r^*| + |\mathcal{C}_i^h[r+1:i]|.$$

□

The remainder of this section shows that if either the $(s,k,\mathcal{I})$-settlement or the $(s,k,\mathcal{I})$-intensive chain quality is violated, then the event $\mathcal{E}^c_{\mathsf{secure}}$ must occur, where

$$\begin{aligned}
\mathcal{E}_{\mathsf{secure}} &= \{N_{\mathsf{adv}}[r+1:i] < N_{\mathsf{spl}}[r+1:i] - \mathsf{Unheard}_{\mathcal{I}}[i] \quad \forall\ r \le s,\ i \ge s+k,\} \\
\mathcal{E}^c_{\mathsf{secure}} &= \{N_{\mathsf{adv}}[r+1:i] \ge N_{\mathsf{spl}}[r+1:i] - \mathsf{Unheard}_{\mathcal{I}}[i] \quad \text{for some } r \le s,\ i \ge s+k.\}
\end{aligned} \tag{2.7}$$

The analysis for the settlement property is given in Section 2.5.1 and that for the chain quality property is given in Section 2.5.2. Both analyses have a similar outline. First, the chain growth properties (Lemmas 2.3 and 2.4) are used to lower bound the length of a certain portion of a chain, namely $\mathcal{C}[r+1:i]$. Next, the fact that the settlement (or chain quality) property is violated is used to upper bound the same quantity with the number of adversarial blocks mined in that interval, $N_{\mathsf{adv}}[r+1:i]$. These bounds give us the lower bound on $N_{\mathsf{adv}}[r+1:i]$ as stated in (2.7).

2.5.1 Settlement Violation Condition

Recall, from Definition 2.1, that the settlement property with parameters $s, k \in \mathbb{N}$ and $\mathcal{I} \subseteq \mathcal{H}$ holds in an execution if the following event occurs:

$$\mathcal{E}_{\mathsf{settlement}} \triangleq \{\forall\ h, h' \in \mathcal{I},\ \forall i, i' \ge s+k, \mathcal{C}_i^h[1:s] = \mathcal{C}_{i'}^{h'}[1:s]\}.$$

As such, if the property is violated, it implies that at two (possibly) different points in time (i, i'), two honest users held different chains. It is more convenient to argue in terms of events that are more local in time. To this end, define the event:

$$\mathcal{E}_{i-\mathsf{settlement}} \triangleq \{\forall\ h, h' \in \mathcal{I},\ \mathcal{C}_i^h[1:s] = \mathcal{C}_i^{h'}[1:s]\} \cap \{\forall\ h \in \mathcal{I}, \mathcal{C}_i^h[1:s] = \mathcal{C}_{i+1}^h[1:s]\} \tag{2.8}$$

From (2.8), it follows that

$$\begin{aligned}
\mathcal{E}^c_{i-\mathsf{settlement}} = \{&\exists\ h, h' \in \mathcal{I} \text{ such that } \mathcal{C}_i^h[1:s] \ne \mathcal{C}_i^{h'}[1:s]\} \\
&\cup \{\exists\ h \in \mathcal{I} \text{ such that } \mathcal{C}_i^h[1:s] \ne \mathcal{C}_{i+1}^h[1:s].\}
\end{aligned} \tag{2.9}$$

$\mathcal{E}_{i-\mathsf{settlement}}$ states that honest nodes agree with each other at slot i, and with themselves across slots i and $i+1$. If $\mathcal{E}_{i-\mathsf{settlement}}$ occurs for all slots $i \ge s+k$, then all nodes in $\mathcal{I}$ agree with each

other from slot $s+k$ onward. This can be shown by induction on i. Therefore,

$$\mathcal{E}_{\mathsf{settlement}} = \bigcap_{i \geq s+k} \mathcal{E}_{i-\mathsf{settlement}}, \quad \text{or equivalently,} \quad \mathcal{E}^c_{\mathsf{settlement}} = \bigcup_{i \geq s+k} \mathcal{E}^c_{i-\mathsf{settlement}}. \tag{2.10}$$

We now have all the ingredients to prove a necessary condition for settlement violation.

Lemma 2.5 (Settlement Violation-Necessary Condition)**.** *Suppose, in an execution, the settlement property with parameters $(s, k, \mathcal{I})$ is violated (i.e., $\mathcal{E}^c_{\mathsf{settlement}}$ occurs). Then, for some $r < s$ and $i \geq s+k$,*

$$N_{\mathsf{adv}}[r+1:i] \geq N_{\mathsf{spl}}[r+1:i] - \mathsf{Unheard}_{\mathcal{I}}[i] \tag{2.11}$$

which implies that $\mathcal{E}^c_{\mathsf{secure}}$ occurs.

Proof. Suppose $\mathcal{E}^c_{\mathsf{settlement}}$ occurs. Then, by (2.10), there exists $i \geq s+k$ such that $\mathcal{E}^c_{i-\mathsf{settlement}}$ occurs. Therefore, it suffices to show that the event $\mathcal{E}^c_{i-\mathsf{settlement}}$ implies (2.11) for some $r < s$. By (2.9), $\mathcal{E}^c_{i-\mathsf{settlement}}$ implies one of two events: either there are two different honest users that have disjoint chains at slot i, or the same honest user has disjoint chains at slot i and $i+1$. The arguments for both cases are presented together, as they are very similar.

In the first case, let $\mathcal{C}_1 \triangleq C^h_i$, $\mathcal{C}_2 \triangleq C^{h'}_i$, and assume, without loss of generality, that $|\mathcal{C}_2| \geq |\mathcal{C}_1|$. In the second case, let $\mathcal{C}_1$ be C^h_i and $\mathcal{C}_2$ be $C^h_{i+1}[1:i]$. Here too, $|\mathcal{C}_2| \geq |\mathcal{C}_1|$, as the following argument shows. In slot $i+1$, party h must have adopted a new chain; otherwise, settlement would not be violated. This new chain (C^h_{i+1}) must be strictly longer than the chain it held at slot i (C^h_i, i.e., C_1). Since $\mathcal{C}_2$ is shorter than C^h_{i+1} by at most one block, it follows that $|\mathcal{C}_2| \geq |\mathcal{C}_1|$.

In both cases, $\mathcal{C}_1$ and $\mathcal{C}_2$ are disjoint from slot s or before. Put differently, all blocks on the common prefix of $\mathcal{C}_1$ and $\mathcal{C}_2$ are mined strictly before slot s. Let r be the slot of the last special honest block in the common prefix of $\mathcal{C}_1$ and $\mathcal{C}_2$. Similarly, define r' be the slot of the last block in the common prefix of $\mathcal{C}_1$ and $\mathcal{C}_2$. In general, $r \leq r'$, with equality holding if and only if the last common block of $\mathcal{C}_1$ and $\mathcal{C}_2$ is special honest.

Keeping the definitions of r and r' in mind, note that $\mathcal{C}_1[r+1:r']$ contains only adversarial blocks, which implies $|\mathcal{C}_1[r+1:r']| \leq N_{\mathsf{adv}}[r+1:r']$. Moreover, observe that for every block on $\mathcal{C}_1[r'+1:i]$, there is a different block on $\mathcal{C}_2$ at the same height. Consider a pair of blocks at the same height, one each on $\mathcal{C}_1$ and $\mathcal{C}_2$. Between these blocks, at least one must be adversarial, because two special honest blocks can never be at the same height on the blocktree. Therefore, $|\mathcal{C}_1[r'+1:i]| \leq N_{\mathsf{adv}}[r'+1:i]$. Putting these inequalities together gives:

$$|\mathcal{C}_1[r+1:i]| = |\mathcal{C}_1[r+1:r']| + |\mathcal{C}_1[r'+1:i]| \leq N_{\mathsf{adv}}[r+1:r'] + N_{\mathsf{adv}}[r'+1:i] = N_{\mathsf{adv}}[r+1:i].$$

Recall that for any $h \in \mathcal{I}$, $\mathsf{Unheard}_{\mathcal{I}}[i] \geq \mathsf{Unheard}_h[i]$. Using this fact along with Lemma 2.4 gives a lower bound for $|\mathcal{C}_1[r+1:i]|$:

$$|\mathcal{C}_1[r+1:i]| \geq N_{\mathsf{spl}}[r+1:i] - \mathsf{Unheard}_h[i] \geq N_{\mathsf{spl}}[r+1:i] - \mathsf{Unheard}_{\mathcal{I}}[i].$$

Together, the upper and lower bounds for $|\mathcal{C}_1[r+1:i]|$ lead to (2.11):

$$N_{\mathsf{adv}}[r+1:i] \geq |\mathcal{C}_1[r+1:i]| \geq N_{\mathsf{spl}}[r+1:i] - \mathsf{Unheard}_{\mathcal{I}}[i].$$

We have proven that settlement violation implies (2.11) holds for some $r < s$ and $i \geq s+k$. This, in turn, implies $\mathcal{E}^c_{\mathsf{secure}}$ occurs. In defining $\mathcal{E}^c_{\mathsf{secure}}$, we have allowed for the possibility of $r = s$ as well, so as to include chain quality violation within the same event. □

2.5.2 Chain Quality Violation

The necessary condition for the chain quality property to be violated is identical to that of the settlement property begin violated, and the analysis has a similar flavor as well.

Lemma 2.6 (Intensive Chain Quality Violation – Necessary Condition)**.** *Suppose, in an execution, the $(s,k,\mathcal{I})$-intensive chain quality property is violated. Then, for some $r \leq s$ and $i \geq s+k$,*

$$N_{\mathsf{adv}}[r+1:i] \geq N_{\mathsf{spl}}[r+1:i] - \mathsf{Unheard}_{\mathcal{I}}[i],$$

i.e., $\mathcal{E}^c_{\mathsf{secure}}$ occurs.

Proof. Consider an execution where the intensive chain quality with parameters s, k and $\mathcal{I}$ is violated. This means that for some honest party $h \in \mathcal{I}$ and $i' \geq s+k$, there are no special honest slots in $\mathcal{C}^h_{i'}[s+1:s+k]$. Let $r \leq s, i \geq s+k$ be chosen such that $\mathcal{C}^h_{i'}[r+1:i]$ is a maximal portion of the chain $\mathcal{C}^h_{i'}$ without any special honest blocks. Thus, r is the slot of the last special honest block on the portion of the chain $\mathcal{C}^h_{i'}[1:s]$, or 0 if there is no such block. In either case, $\mathcal{C}^h_i[1:r] = \mathcal{C}^*_r$. Similarly, if the portion of the chain $\mathcal{C}^h_{i'}[s+k+1:i']$ contains at least one special honest block, then $i+1$ is the slot of the first such block; else, $i = i'$. We know that all blocks in $\mathcal{C}^h_{i'}[r+1:i]$ must be adversarial blocks, mined in the interval $\{r+1,\ldots,i\}$. Therefore, $|\mathcal{C}^h_{i'}[r+1:i]| \leq N_{\mathsf{adv}}[r+1:i]$. To prove the lemma, it thus suffices to show:

$$|\mathcal{C}^h_{i'}[r+1:i]| \geq N_{\mathsf{spl}}[r+1:i] - \mathsf{Unheard}_h[i]. \tag{2.12}$$

The proof of (2.12) is divided into the cases $i < i'$ and $i = i'$.

($i < i'$). In this case, we know that $i+1$ is a special honest slot, and the block mined in that slot is part of the chain $\mathcal{C}^h_{i'}$. This implies the following facts:

- $\mathcal{C}^h_{i'}[1:i+1] = \mathcal{C}^*_{i+1}$. Moreover, $|\mathcal{C}^h_{i'}[1:i]| = |\mathcal{C}^*_{i+1}| - 1$.
- $N_{\mathsf{spl}}[r+1:i+1] - 1 = N_{\mathsf{spl}}[r+1:i]$.

Also note that $\mathcal{C}^h_{i'}[1:r] = \mathcal{C}^*_r$. Using these facts, we get $|\mathcal{C}^h_{i'}[r+1:i]| = |\mathcal{C}^h_{i'}[1:i]| - |\mathcal{C}^h_{i'}[1:r]|$ $= |\mathcal{C}^*_{i+1}| - 1 - |\mathcal{C}^*_r| \geq N_{\mathsf{spl}}[r+1:i+1] - 1 = N_{\mathsf{spl}}[r+1:i]$. The previous inequality follows from Lemma 2.3. This implies (2.12), as $\mathsf{Unheard}_h[i]$ is always nonnegative.

($i = i'$) In this case, (2.12) follows immediately from the second statement of Lemma 2.4. □

2.5.3 Reach

We conclude this section by introducing the notion of Reach, which allows us to write a reformulate $\mathcal{E}^c_{\mathsf{secure}}$ that is easier to analyze.

Definition 2.7 (Reach)**.** *Let* $\mathsf{Reach}[0] = 0$*, and for any* $i \in \mathbb{N}$*, let*

$$\mathsf{Reach}[i] = \max_{j:0\le j\le i} N_{\mathsf{adv}}[j+1:i] - N_{\mathsf{spl}}[j+1:i]$$

In the above definition, $N_{\mathsf{adv}}[i+1:i], N_{\mathsf{spl}}[i+1:i]$ are zero, since the interval $[i+1:i]$ is empty. Thus, Reach is a random process that is a function of $\mathsf{CharString}$ and it takes values in $\mathbb{Z}_+$.

The Ouroboros line of works [20, 21, 22, 25] define $\mathsf{Reach}[i]$ as an upper bound on the extent to which an adversarial chain can 'reach' beyond any honest chain in the block-tree $\mathcal{F}_i$. The two definitions can be shown to be equivalent. With this definition in place, we can rewrite $\mathcal{E}^c_{\mathsf{secure}}$ as:

$$\mathcal{E}^c_{\mathsf{secure}} = \{N_{\mathsf{adv}}[s+1:i] - N_{\mathsf{spl}}[s+1:i] + \mathsf{Unheard}_{\mathcal{I}}[i] + \mathsf{Reach}[s] \ge 0 \text{ for some } i \ge s+k\}. \quad (2.13)$$

This can be proved by first splitting $N_{\mathsf{spl}}[r+1:i]$ as $(N_{\mathsf{spl}}[r+1:s] + N_{\mathsf{spl}}(\mathsf{CharString}[s+1:i])$ (and likewise for $N_{\mathsf{adv}}[r+1:i]$), and then noting that for any value x,

$$\{N_{\mathsf{adv}}[r+1:s] - N_{\mathsf{spl}}[r+1:s] \ge x \text{ for some } r \le s\} \Leftrightarrow \{\mathsf{Reach}[s] \ge x\}.$$

The remaining chapter is devoted to finding a bound on $\mathbb{P}(\mathcal{E}^c_{\mathsf{secure}})$.

2.6 Lemmas on Probabilistic Properties

Consider the expression of $\mathcal{E}^c_{\mathsf{secure}}$ in (2.13). To bound $\mathbb{P}(\mathcal{E}^c_{\mathsf{secure}})$, we need to bound the three terms $N_{\mathsf{adv}}[s+1:i] - N_{\mathsf{spl}}[s+1:i]$, $\mathsf{Unheard}_{\mathcal{I}[i]}$ and $\mathsf{Reach}[s]$. The first two terms should be viewed as processes, i.e., functions of i. These processes need to be bounded for all time (beyond slot $s+k$). To obtain such bounds, we work with compressed versions of these processes, obtained by sampling them at nonempty slots alone. Section 2.6.1 defines the necessary terms and results to perform this mapping between time scales. Sections 2.6.2, 2.6.3, and 2.6.4 analyze the distribution of the processes $\mathsf{CharString}$, $\mathsf{Unheard}$, and Reach respectively. The results of this section are combined to give an upper bound on $\mathbb{P}(\mathcal{E}^c_{\mathsf{secure}})$ in Section 2.7.

2.6.1 Reduction to compressed time scale

Define a new time scale driven by the arrival of nonempty slots: the clock ticks by one whenever a new nonempty slot occurs. Call this event-driven time scale the *compressed time scale*. The following notation is used to define the processes $\mathsf{CharString}$ and $\mathsf{Unheard}$ on the compressed time

scale. For $s \geq 0$ and $j \geq 1$, let

$$T_j^s = \min\{i : N[s+1 : s+i] = j\} \tag{2.14}$$

In other words, T_j^s is the j^{th} nonempty slot strictly after time s. Clearly, $T_j^0 = T_j$. Note that, for any $s \geq 0$, T_1^s and the random variables $\{T_j^s - T_{j-1}^s\}_{j \geq 2}$ are i.i.d. with distribution $\mathsf{geom}(f)$. Let $\mathsf{CompressedCharString}_s$ denote the time-shifted, compressed version of relative to reference slot s:

$$\begin{aligned} \mathsf{CompressedCharString}_s[0] &= \mathsf{CharString}[s], \\ \mathsf{CompressedCharString}_s[j] &= \mathsf{CharString}[s + T_j^s] \text{ for } j \geq 1. \end{aligned} \tag{2.15}$$

$\mathsf{CompressedUnheard}_{h,s}$ and $\mathsf{CompressedUnheard}_{\mathcal{I},s}$, which are the compressed versions of $\mathsf{Unheard}_h$ and $\mathsf{Unheard}_{\mathcal{I}}$ respectively, are also defined in a similar manner. Finally, let N_{spl}^s and N_{adv}^s be time-shifted, compressed versions of N_{spl} and N_{adv} respectively. Specifically,

$$N_{\mathsf{spl}}^s[1:j] = N_{\mathsf{spl}}[s + T_1^s : s + T_j^s] = N_{\mathsf{spl}}[s+1 : s + T_j^s].$$

$N_{\mathsf{adv}}^s[1:j]$ is defined similarly. We now state the main result of this subsection.

Lemma 2.7. *Given $k' \geq 1$, let:*

$$\begin{aligned} &F_0 = \{T_{k'}^s > k\} \text{ and} \\ &F_1 = \{N_{\mathsf{adv}}^s[1:j] - N_{\mathsf{spl}}^s[1:j] + \mathsf{CompressedUnheard}_{\mathcal{I},s}[j] + \mathsf{Reach}[s] \geq 0 \text{ for some } j \geq k'\}. \end{aligned}$$

Then, the event $\mathcal{E}_{\mathsf{secure}}^c$ satisfies $\mathcal{E}_{\mathsf{secure}}^c \subseteq F_0 \cup F_1$, which implies $\mathbb{P}(\mathcal{E}_{\mathsf{secure}}^c) \leq \mathbb{P}(F_0) + \mathbb{P}(F_1)$.

Proof. First, note that both $N_{\mathsf{adv}}[s+1:i]$ and $N_{\mathsf{spl}}[s+1:i]$, as functions of i, are constant over intervals of the form $[T_j^s : T_{j+1}^s - 1]$. Second, the process $\mathsf{Unheard}_{\mathcal{I}}[i]$ is nonincreasing over such intervals. This is because $\mathsf{Unheard}_{\mathcal{I}}[i]$ only increases at special honest slots. Now, suppose the event $\mathcal{E}_{\mathsf{secure}}^c$ holds. If j is such that $s + T_j^s$ is the last renewal time less than or equal to i, then

$$N_{\mathsf{adv}}^s[1:j] - N_{\mathsf{spl}}^s[1:j] + \mathsf{CompressedUnheard}_{\mathcal{I},s}[j] + \mathsf{Reach}[s] \geq 0.$$

If F_0 does not hold, then $s + T_{k'}^s \leq s + k$. This implies that $j \geq k'$, and hence F_1 is true. This completes the proof of the first claim. This claim implies that $\mathbb{P}(\mathcal{E}_{\mathsf{secure}}^c)$ is bounded above by $\mathbb{P}(F_0 \cup F_1)$. This, by the union bound, is bounded above by $\mathbb{P}(F_0) + \mathbb{P}(F_1)$. This shows the second claim. □

The following lemma gives a bound on $\mathbb{P}(F_0) = \mathbb{P}(T_{k'}^s > k)$.

Lemma 2.8. *Suppose $k' = \lceil rkf \rceil$ such that $0 < r < 1$. Then $\mathbb{P}(T_{k'}^s > k) \leq \exp(-kf(1-r)^2/2)$.*

Proof. Note that $\{T_{k'}^s > k\} = \{N[s+1 : s+k] \leq k' - 1\}$, and $N[s+1 : s+k]$ has the binomial

distribution with parameters k and f. Thus

$$\begin{aligned}\mathbb{P}(T^s_{k'} > k) &= \mathbb{P}(\mathsf{binom}(k, f) \leq k' - 1) \\ &\leq \mathbb{P}(\mathsf{binom}(k, f) \leq rkf) \leq \exp(-kf(1-r)^2/2).\end{aligned}$$

Here, we use the bound $\mathbb{P}(\mathsf{binom}(n, p) \leq rnp) \leq \exp(np(1-r)^2/2)$. □

The results in the following subsections are used to bound $\mathbb{P}(F_1)$.

2.6.2 On CharString

The main result of this subsection, Lemma 2.9, characterizes the distribution of CharString. In essence, it specifies how to construct a process with identical distribution to CharString. First, generate a stationary renewal process on $\mathbb{Z}$ with lifetime distribution $\mathsf{geom}(f)$. At all slots except the renewal points, set the symbol to $\perp$. Let the renewal points be indexed by $j \in \mathbb{Z}$. At each renewal point j, set the symbol to 0 with probability $\alpha\mathbb{P}(\Delta < g_j)$ and to 1 otherwise, and do so independently for every renewal point; here, g_j is the gap between the j^{th} and $(j-1)^{\text{th}}$ renewal point. Here, we give the proof for the one-time leader model. The identical statement for the i.i.d. leader model is proven in Appendix 2.8.

Lemma 2.9 (The Distribution of CharString)**.** *The sequence of nonempty slots,* $(T_j : j \in \mathbb{Z})$*, forms a stationary renewal process with lifetime distribution* $\mathsf{geom}(f)$*. Conditioned on* $(T_j : j \in \mathbb{Z})$*, the random variables* $(\mathsf{CharString}[T_j] : j \in \mathbb{Z})$ *are independent and Bernoulli with the following distribution:*

$$\mathbb{P}(\mathsf{CharString}[T_j] = 0 | T_{j'} : j' \in \mathbb{Z}) = \alpha\mathbb{P}(\Delta < T_j - T_{j-1} | T_j - T_{j-1}) \quad \forall\ j \in \mathbb{Z}.$$

In particular, CharString *is a stationary process.*

Proof. The first claim is already established in Section 2.4.1; it follows from the distribution of LeaderString. The second claim is true for $j \leq 0$ by construction, as shown in Section 2.4.2. We now show the second claim for $j \geq 1$. Recall, from Section 2.4.2, $\mathsf{CharString}[T_j] = 0$ if $\mathsf{LeaderString}[T_j] = 0$ and $R_j < T_j - T_{j-1}$, where R_j is the delay from the last special honest slot before T_j to h_j. The sequence $(\mathsf{LeaderString}[T_j] : j \in \mathbb{Z})$ is a Bernoulli process with probability of zero being α. Moreover, it is independent of the sequence $(T_j : j \in \mathbb{Z})$. It therefore suffices to show that given LeaderString, the sequence of random variables $(R_j : j \in \mathbb{N}, \mathsf{LeaderString}[T_j] = 0)$ are i.i.d. with the same distribution as Δ.

The last statement can be shown in three steps. First, given LeaderString, the message delays are mutually independent with the same distribution as Δ. Second, the sequence $(R_j : j \in \mathbb{N}, \mathsf{LeaderString}[T_j] = 0)$ is a subset of all the delay random variables, and whether a delay variable is in this sequence or not is independent of its value. Third, as a consequence of the one-time leader

model, each R_j represents a different message's delay. This is because the recipient h_j is always different, even if the sender h_{j^*} may be the same for different j's.

The fact that the nonempty slots form a stationary process with independent lifetimes, together with the fact that the value of $\mathsf{CharString}$ at the nonempty slots is determined by the immediate lifetime, implies that $\mathsf{CharString}$ is a stationary process. □

From Lemma 2.9, we get the following result, which states that the process $\mathsf{CompressedCharString}_s$ is nearly a Bernoulli process, and in fact, can be stochastically dominated by one.

Lemma 2.10 (The Distribution of $\mathsf{CompressedCharString}$)**.** *For any $s \geq 0$,*

$$\mathbb{P}(\mathsf{CompressedCharString}_s[1] = 0 | \mathsf{CharString}[:s]) \geq p.$$

Further, for any $s \geq 0$ and $j \geq 2$,

$$\mathbb{P}(\mathsf{CompressedCharString}_s[j] = 0 | \mathsf{CharString}[:s + T^s_{j-1}]) = p.$$

Here, $p = \alpha\mathbb{P}(\Delta < G)$ is the parameter defined in Definition 2.5, and $\mathsf{CharString}[:s]$ denotes the portion of $\mathsf{CharString}$ indexed by $j \in \mathbb{Z} : j \leq s$.

Proof. By the stationarity of $\mathsf{CharString}$ (from Lemma 2.9), it suffices to prove the lemma for $s = 0$. The desired result follows from Lemma 2.9 and carefully arguing about conditional probabilities.

Pick any $j \in \mathbb{N}$. Lemma 2.9 implies the following: if we condition on $(T_{j'} : j' \leq j)$ and $(\mathsf{CharString}[T_{j'}] : j' \leq j-1)$, the conditional probability of $\mathsf{CharString}[T_j] = 0$ is $\alpha\mathbb{P}(\Delta < T_j - T_{j-1} | T_j - T_{j-1})$. Now consider taking out T_j from the conditioned variables. In other words, restrict the information being conditioned on to $(T_{j'} : j' \leq j-1)$ and $(\mathsf{CharString}[T_{j'}] : j' \leq j-1)$. The conditional probability of $\mathsf{CharString}[T_j] = 0$ is now $\alpha\mathbb{P}(\Delta < T_j - T_{j-1})$, where $T_j - T_{j-1}$ is independent of Δ.

If $j \geq 2$, $T_j - T_{j-1}$ has the lifetime distribution, $\mathsf{geom}(f)$. Thus, the conditional probability above is simply p. For $j = 1$, $T_1 - T_0$ has the sampled lifetime distribution, which is the sum of two independent $\mathsf{geom}(f)$ random variables minus one (see Section 2.4.1). This distribution stochastically dominates the $\mathsf{geom}(f)$ distribution. Therefore, the conditional probability above is lower bounded by p. Finally, note that $\mathsf{CharString}[:T_{j-1}]$ is a deterministic function of the sequences $(T_{j'} : j' \leq j-1)$ and $(\mathsf{CharString}[T_{j'}] : j' \leq j-1)$. Therefore, the statements of the lemma follow. □

The last result of this subsection gives a bound on the process $(N^s_{\mathsf{adv}}[1:j] - N^s_{\mathsf{spl}}[1:j], j \geq 1)$. It states that the process eventually remains below a line whose slope is greater than its drift.

Lemma 2.11. *For any $c < \epsilon$,*

$$\mathbb{P}(N^s_{\mathsf{adv}}[1:j] - N^s_{\mathsf{spl}}[1:j]) \geq -cj \text{ for some } j \geq k') \leq 2\exp(-k'(\epsilon - c)^2/3)$$

Proof. Let W denote a simple integer valued random walk with a drift $-\epsilon$, where $0 < \epsilon < 1$. In other words, $\mathsf{W}[0] = 0$ and

$$\mathsf{W}[j+1] = \begin{cases} \mathsf{W}[j] + 1 & \text{w.p. } \frac{1-\epsilon}{2} \\ \mathsf{W}[j] - 1 & \text{w.p. } \frac{1+\epsilon}{2} \end{cases} \tag{2.16}$$

Note that, for any $s \in \mathbb{N}$, Lemma 2.10 implies

$$(N^s_{\mathsf{adv}}[2:j] - N^s_{\mathsf{spl}}[2:j], j \geq 2) \stackrel{d.}{=} (\mathsf{W}[j] - \mathsf{W}[1], j \geq 2).$$

Moreover, the same lemma implies $(N^s_{\mathsf{adv}}[1:j] - N^s_{\mathsf{spl}}[1:j], j \geq 1)$ is stochastically dominated by $(\mathsf{W}[j], j \geq 1)$, because the former has value zero at $j = 1$ with probability greater than p. Thus, for this proof, it suffices to show that the same bound holds for $(\mathsf{W}[j], j \geq 1)$ instead of $(N^s_{\mathsf{adv}}[1:j] - N^s_{\mathsf{spl}}[1:j], j \geq 1)$. For any $c < \epsilon$, for any $k \in \mathbb{N}$,

$$\mathbb{P}(W[j] \geq -cj \text{ for some } j \geq k) \leq 2\exp\left(-k(\epsilon - c)^2/3\right) \tag{2.17}$$

Let $b > 0$, to be determined below. Observe that the event on the left-hand side of (2.17) is contained in $G_1 \cup G_2$ where $G_1 = \{W[k] \geq -ck - b\}$ and $G_2 = \{\max_{i \geq 0}(W[i+k] - W[k] + ci) \geq b\}$.

Since $W[k] + k\epsilon$ is the sum of k i.i.d. random variables with 0 mean, each taking values in an interval of length two, Hoeffding's inequality implies that for any $\delta > 0$, $\mathbb{P}\left(W[k] + k\epsilon \geq k\delta\right) \leq \exp(-k\delta^2/2)$. Setting $\delta = \epsilon - c - (b/k)$ yields $P(G_1) \leq \exp(-k\delta^2/2)$.

Let Y be a random variable such that

$$Y = \begin{cases} 1 + c & \text{w.p. } \frac{1-\epsilon}{2} \\ -1 + c & \text{w.p. } \frac{1+\epsilon}{2}, \end{cases}$$

and let $Y_1, Y_2, \ldots$ be i.i.d. copies of Y. Kingman's tail bound [31] is that, for $\theta^* = \sup\{\theta > 0 : \mathbb{E}\left[e^{\theta Y}\right] \leq 1\}$,

$$\mathbb{P}\left(\max_{i \geq 0} \sum_{i'=1}^{i} Y_{i'} \geq b\right) \leq e^{-\theta^* b}$$

To obtain a bound on θ^*, note that Hoeffding's lemma for bounded random variables [32] implies that $\mathbb{E}\left[e^{\theta(Y - (c-\epsilon))}\right] \leq e^{\theta^2/2}$. Taking $\theta = -2(c - \epsilon)$ shows that $\mathbb{E}\left[e^{2(\epsilon - c)Y}\right] \leq 1$. Therefore $\theta^* \geq 2(\epsilon - c)$, Thus, for any $b \geq 0$,

$$\mathbb{P}\left(\max_{i \geq 0} \sum_{i'=1}^{i} Y_{i'} \geq b\right) \leq e^{-\theta^* b} \leq e^{-2(\epsilon - c)b}, \tag{2.18}$$

For any $k \in \mathbb{N}$, we note that the random processes $(\sum_{i'=1}^{i} Y_{i'} : i \geq 0)$ and $(W[i+k] - W[k] + ci : i \geq 0)$ have the same distribution. Therefore, (2.18) implies $\mathbb{P}(G_2) \leq \exp\left(-2(\epsilon - c)b\right)$.

Thus $\mathbb{P}(G_1 \cup G_2) \leq \exp\left(-k\delta^2/2\right) + \exp\left(-2(\epsilon - c)b\right)$. Setting $b = k(\epsilon - c)\left(1 - \sqrt{\frac{2}{3}}\right)$ gives $\delta^2/2 = (\epsilon - c)^2/3$. Using $2\left(1 - \sqrt{\frac{2}{3}}\right) \geq 1/3$ yields

$$\mathbb{P}(G_1 \cup G_2) \leq 2\exp\left(-k(\epsilon - c)^2/3\right)$$

which proves (2.17), and thus proves the lemma.

□

2.6.3 On $\mathsf{Unheard}$

The main result of this subsection, Lemma 2.12, states that the marginal distribution of both $\mathsf{Unheard}_h$ and $\mathsf{CompressedUnheard}_{h,s}$ are stochastically dominated by a geometric distribution with an appropriate parameter. The statement is easy to foresee through the following understanding. A given honest party h receives a fresh new message at every special honest slot. Therefore, $\mathsf{Unheard}_h[i]$ has the interpretation of the number of consecutive failed trials until the first success; here, each trial corresponds to a different special honest slot and success implies the message from that special honest slot being heard by slot i. These trials are independent. By upper bounding the probability of failure away from one, we obtain a geometric bound on $\mathsf{Unheard}_h[i]$.

Lemma 2.12 (The Distribution of Unheard)**.** *Let $q \triangleq \mathbb{P}(\Delta \leq \mathsf{geom}(f))$. The following statements hold:*

(a) For any $i \geq 1$, $\mathbb{P}(\mathsf{Unheard}_h[i] > a) \leq (1-q)^a$ for all integers $a \geq 0$.

(b) For any $s, j \geq 1$, $\mathbb{P}(\mathsf{CompressedUnheard}_{h,s}[j] > a) \leq (1-q)^a$ for all integers $a \geq 0$.

Proof. Fix $i \geq 1$. It is possible that i itself is a special honest slot and h has not heard it by slot i. In any case, $\mathsf{Unheard}_h[i]$ is less than or equal to one plus the number of consecutive special honest slots from strictly before slot i that h has not heard by slot i. The nonempty slots form both a Bernoulli process with parameter f and a renewal process. Let $D_1, D_2, \ldots$ denote the lifetimes of the renewal process going backwards from slot i. Thus, $i - D_1 - \cdots - D_j$ is the j^{th} nonempty slot before i. The random variables D_i are independent with the $\mathsf{geom}(f)$ distribution.

The last special honest slot before slot i must be at least D_1 slots before slot i, so the probability h has heard that special honest slot is at least q. In general, for $j \geq 1$, the j^{th} from the last special honest slot before slot i must be at least D_j slots before slot i. (Here D_j is used as a lower bound on $D_1 + \cdots + D_j$.) Thus, no matter which of the last $j-1$ special honest slots before slot i that h has heard, the probability h hears the j^{th} from last special honest slot before i is at least q. Therefore, $\mathsf{Unheard}_h[i]$ can be viewed as at most one plus the number of consecutive failures in a sequence of trials, such that each successive trial is successful with probability at least q. Thus, $\mathsf{Unheard}_h[i]$ is stochastically dominated by the $\mathsf{geom}(q)$ distribution, which is the conclusion of (a).

The proof of (b) is obtained in a similar fashion by replacing i with $s + T_j^s$, for a fixed $s \geq 1$ and $j \geq 1$. There is just one subtlety which we note here. If we consider the renewal process from the perspective of slot $s + T_j^s$, which is the j^{th} renewal point after slot s, the j^{th} lifetime going backwards has the sampled lifetime distribution; all the other lifetimes have the $\mathsf{geom}(f)$ distribution. The sampled lifetime distribution is equivalent to the sum of two independent $\mathsf{geom}(f)$ random variables minus one and is stochastically greater that the typical lifetime distribution, $\mathsf{geom}(f)$. Thus, the arguments in the proof of (a) continue to hold. Importantly, these lifetimes are mutually independent. □

The next lemma follows from the above result and the union bound.

Lemma 2.13. *For any $k' \in \mathbb{N}$, $b \geq 0$ and $c > 0$,*

$$\mathbb{P}(\mathsf{CompressedUnheard}_{h,s}[j] \geq b + 1 + c(j - k') \text{ for some } j \geq k') \leq \left[\frac{1}{(1-(1-q)^c)}\right] \exp(-bq).$$

Proof. Lemma 2.12 (b) implies

$$\mathbb{P}(\mathsf{CompressedUnheard}_{h,s}[j] \geq t) \leq (1-q)^{t-1} \text{ for } t \in \mathbb{R}_+.$$

Substituting $b + 1 + c(j - k')$ for t and using the union bound by summing over the possible values of $j - k'$ implies that the left-hand side of (2.13) is bounded from above by

$$\sum_{d=0}^{\infty} (1-q)^{b+cd} = \left[\frac{1}{(1-(1-q)^c)}\right] (1-q)^b.$$

Since $(1-q)^b \leq \exp(-bq)$, the bound in equation (2.13) follows. □

2.6.4 On Reach

In this section, we show that the distribution of Reach has a geometric tail. From Definition 2.7, it follows by induction on i that Reach satisfies the following recursion:

$$\mathsf{Reach}[i] = \begin{cases} \mathsf{Reach}[i-1] & \text{if } \mathsf{CharString}[i] = \perp \\ \mathsf{Reach}[i-1] + 1 & \text{if } \mathsf{CharString}[i] = 1 \\ \max(\mathsf{Reach}[i-1] - 1, 0) & \text{if } \mathsf{CharString}[i] = 0 \end{cases} \tag{2.19}$$

The initial condition of the recursion, $\mathsf{Reach}[0] = 0$, is given in Definition 2.7 itself.

Next, define B to be the backwards residual lifetime process for the locations of the non-empty slots in $\mathsf{CharString}$, counting from zero. Mathematically,

$$\mathsf{B}[i] = \min\{i' \geq 0 : \mathsf{CharString}[i - i'] \neq \perp\}.$$

The next lemma characterizes the joint process $(\mathsf{B}, \mathsf{Reach})$.

Lemma 2.14. *The process* $(\mathsf{B}[i], \mathsf{Reach}[i])$ *is a discrete-time Markov process with equilibrium probability mass function given by*

$$\pi(b, r) = f(1-f)^b \left(1 - \frac{1-p}{p}\right) \left(\frac{1-p}{p}\right)^r.$$

In other words, under the equilibrium distribution, $\mathsf{B}[i]$ *is independent of* $\mathsf{Reach}[i]$*,* $\mathsf{B}[i]$ *has the* $\mathsf{geom}(f) - 1$ *distribution and* $\mathsf{Reach}[i]$ *has the* $\mathsf{geom}\left(\frac{1-p}{p}\right) - 1$ *distribution.*

Proof. The Markov property follows from

1. the recursion (2.19) for determining Reach from $\mathsf{CharString}$ and
2. the renewal structure of $\mathsf{CharString}$ described in Lemma 2.9.

The nonzero transition probabilities out of any given state $(b, r) \in \mathbb{Z}_+^2$ are given by (with $F_b = \mathbb{P}(\Delta \leq b)$):

$$\begin{aligned}
\mathbb{P}((b, r) \to (b+1, r)) &= 1 - f \\
\mathbb{P}((b, r) \to (0, (r-1)_+)) &= f\alpha F_b \\
\mathbb{P}((b, r) \to (0, r+1)) &= f(1 - \alpha F_b)
\end{aligned}$$

To verify π is the equilibrium distribution, it suffices to check that if the state of the process at one time has distribution π, then in one step of the process, the probability of jumping out of any given state is equal to the probability of jumping into the state. For a state of the form (b, r) with $b \geq 1$, the probability of jumping into the state is $\pi(b-1, r)(1-f)$, which is equal to $\pi(b, r)$, the probability of jumping out of the state. For a state of the form $(0, r)$ with $r \geq 1$, the probability of jumping into the state satisfies the following:

$$\begin{aligned}
&\sum_{b=0}^{\infty} \pi(b, r-1) f(1 - \alpha F_b) + \sum_{b=0}^{\infty} \pi(b, r+1) f \alpha F_b \\
&\qquad = \pi(0, r) \left[\sum_{b=0}^{\infty} \frac{p}{1-p} (1-f)^b f(1 - \alpha F_b) + \sum_{b=0}^{\infty} \frac{1-p}{p} (1-f)^b f \alpha F_b \right] \\
&\qquad = \pi(0, r),
\end{aligned}$$

where we used the fact $\alpha \sum_{b=0}^{\infty} (1-f)^b f F_b = p$. Thus, the probability of jumping into the state $(0, r)$ is equal to $\pi(0, r)$, which is the probability of jumping out of state $(0, r)$. It remains to show probabilities of jumping into and out of state $(0, 0)$ are the same, but that follows from the fact it is true for all other states. □

This result gives us a handle on the marginal distribution of $\mathsf{Reach}[i]$, as shown below.

Lemma 2.15 (The Distribution of Reach)**.** *For all* $i \in \mathbb{Z}_+$, $a \in \mathbb{R}_+$, $\mathbb{P}(\mathsf{Reach}[i] \geq a) \leq \left(\frac{1-p}{p}\right)^a$.

Proof. The initialization of Reach is $\mathsf{Reach}[0] = 0$. Consider a comparison system such that $\mathsf{Reach}[0]$ is a random variable independent of $\mathsf{CharString}$ with the $\mathsf{geom}\left(\frac{1-p}{p}\right) - 1$ distribution. Then in the comparison system, $((\mathsf{B}[i], \mathsf{Reach}[0]) : i \geq 0)$ is a stationary Markov process, and in particular, $\mathsf{Reach}[i]$ has the $\mathsf{geom}\left(\frac{1-p}{p}\right) - 1$ distribution for all i. Now note that, for $\mathsf{CharString}$ fixed, all the variables $((\mathsf{B}[i], \mathsf{Reach}[i]) : t \geq 0)$ are nondecreasing functions of the initial state $(B[0], \mathsf{Reach}[0])$, as can be readily shown by induction on i. Since the actual initial state of the original system is less than the initial state of the comparison system, it follows that $\mathsf{Reach}[i]$ in the original system is stochastically dominated by the $\mathsf{geom}\left(\frac{1-p}{p}\right) - 1$ distribution. □

2.7 Proof of Theorem 2.1

Theorem 2.1 provides an upper bound on the probability that either settlement or intensive chain quality is violated in an execution. In Section 2.5, we showed that if either security property is violated, then $\mathcal{E}^c_{\mathsf{secure}}$ must necessarily occur. Further, Lemma 2.7 shows that $\mathbb{P}\left(\mathcal{E}^c_{\mathsf{secure}}\right) \leq \mathbb{P}(F_0) + \mathbb{P}(F_1)$ for appropriately defined events F_0 and F_1. Thus, to prove Theorem 2.1, it suffices to show

$$\mathbb{P}(F_0) + \mathbb{P}(F_1) \leq p_{\mathsf{char\text{-}string}} + |\mathcal{I}| p_{\mathsf{unheard}}. \tag{2.20}$$

By Lemma 2.8, for any $r \in (0, 1)$,

$$\mathbb{P}(F_0) \leq \exp(-kf(1-r)^2/2) \leq \exp(-kf\epsilon^2(1-r)^2/2). \tag{2.21}$$

Recall from Lemma 2.7 that

$$F_1 = \{N^s_{\mathsf{adv}}[1:j] - N^s_{\mathsf{spl}}[1:j] + \mathsf{CompressedUnheard}_{\mathcal{I},s}[j] + \mathsf{Reach}[s] \geq 0 \text{ for some } j \geq k'\}.$$

Consider the following bounds on the terms that appear in the expression of F_1 above:

$$\mathbb{P}(N^s_{\mathsf{adv}}[1:j] - N^s_{\mathsf{spl}}[1:j]) \geq -\epsilon cj \text{ for some } j \geq k') \leq 2\exp(-k'\epsilon^2(1-c)^2/3), \tag{2.22}$$

$$\mathbb{P}(\mathsf{CompressedUnheard}_{h,s}[j] \geq \epsilon c(j-k') + \epsilon(c-a)k' + 1 \text{ for some } j \geq k') \leq \left[\frac{1}{1-(1-q)^{\epsilon c}}\right] \exp(-(c-a)q\epsilon k'), \tag{2.23}$$

$$\mathbb{P}(\mathsf{Reach}[s] \geq \epsilon a k') \leq \exp(-2a\epsilon^2 k'). \tag{2.24}$$

The first bound follows from Lemma 2.11 (replace c by ϵc), the second from Lemma 2.13 (replace b by $\epsilon(c-a)k'$ and c by ϵc), and the third from Lemma 2.15 (replace a by $\epsilon a k'$ and use the fact that $\log((1+\epsilon)/(1-\epsilon)) \geq 2\epsilon$). Here, a, r, and c are free parameters belonging to the set $(0, 1)$,

while $k' = \lceil rkf \rceil$. By the union bound, (2.22), (2.23), and (2.24) give the following result:

$$\mathbb{P}(F_1) \leq 2\exp\left(-\frac{(1-c)^2}{3}\epsilon^2 k'\right) + \exp(-2a\epsilon^2 k') + \left[\frac{|\mathcal{I}|}{1-(1-q)^{\epsilon c}}\right]\exp(-(c-a)q\epsilon k'). \tag{2.25}$$

In obtaining this bound, we use the following fact: for three discrete terms x, y, z, the condition $x+y+z \geq 0$ implies either $x \geq 0$, $y \geq 0$, or $z \geq 1$. We also use the fact that $\mathsf{CompressedUnheard}_{\mathcal{I},s}$ is the maximum of $\mathsf{CompressedUnheard}_{h,s}$ for all $h \in \mathcal{I}$.

With bounds on F_0 and F_1 (namely, (2.21) and (2.25)), it remains to set the free variables. Let $r = 1/2, c = 1/4$ and $a = 1/8$. This implies $k' \geq fk/2$. Also use the fact that $q \geq p \geq 0.5$ (see Lemma 2.12 for q). Lastly, use the bound $\left[\frac{1}{1-(1/2)^{\epsilon/4}}\right] \leq (8/\epsilon)$, which follows from $\frac{1}{1-(1-x)^y} \leq 1/xy$ for $x, y < 1$. The above steps show that (2.20) holds for

$$p_{\mathsf{char\text{-}string}} = 4\exp\left(-(3/32)\epsilon^2 fk\right) \quad \text{and} \quad p_{\mathsf{unheard}} = (8/\epsilon)\exp(-(1/32)\epsilon fk).$$

This concludes the proof of Theorem 2.1.

2.8 CharString for the I.I.D. Leader Model

In this section, we give the precise construction of CharString for the i.i.d. leader model, show that it implies the property of increasing heights of special honest blocks, and prove Lemma 2.9 for this construction. All other properties of CharString follow from these. Before we give the precise construction, we need to develop some notation.

2.8.1 Internal representation and refreshed residuals

The definitions in this section are used in the following to define special honest slots for the i.i.d. leader model. Consider a probability mass function (pmf) f on $\mathbb{Z}_+$, and let X be a random variable with this pmf ($\mathbb{P}(X = i) = \mathsf{f}[i]$). The *failure rate function* of the distribution, FailureRate, is defined by

$$\mathsf{FailureRate}[i] \triangleq \frac{\mathsf{f}[i]}{\sum_{j \geq i}\mathsf{f}[i]} = \frac{\mathbb{P}(X = i)}{\mathbb{P}(X \geq i)} \quad \text{for each } i \geq 0$$

with the convention that $\mathsf{FailureRate}[i] = 1$ if $\mathbb{P}(X \geq i) = 0$.

A random variable D with pmf f can be constructed as follows. Let $D = \min\{i \geq 0 : \mathsf{U}[i] \leq \mathsf{FailureRate}[i]\}$, where $\mathsf{U} = (\mathsf{U}[0], \mathsf{U}[1], \ldots)$ is a sequence of independent random variables that are each uniformly distributed on the interval $[0, 1]$. We call $(\mathsf{FailureRate}, \mathsf{U})$ the *internal representation* of D. If D_1 and D_2 are random variables with independent internal representations, then D_1 and D_2 are independent as well.

Given $d \geq 0$, define the *refreshed residual* of D at elapsed time d by $\mathsf{refresh}_d(D) = \min\{i \geq 0 : \mathsf{U}[i+d] \leq \mathsf{FailureRate}[i]\}$. Although $\mathsf{refresh}_d(D)$ depends on the internal representation of D, the

internal representation is suppressed in the notation.

Lemma 2.16. *Let D be a $\mathbb{Z}_+$-valued random variable with an internal representation and let $d \geq 0$. The following hold.*

(a) $\mathsf{refresh}_d(D) \stackrel{d.}{=} D$.

(b) The random variable $\min\{d, D\}$ *is independent of* $\mathsf{refresh}_d(D)$. *More generally, if* $0 = d_0 < d_1 < \cdots < d_n$, *then* $\forall j \in [n]$, $\min\{d_j, \mathsf{refresh}_{d_{j-1}}(D)\}$ *and* $\mathsf{refresh}_{d_n}(D)$ *are mutually independent.*

(c) If D has a non-decreasing failure rate function, $D \leq d + \mathsf{refresh}_d(D)$.

Proof. Statement (a) follows from $\mathsf{U} \stackrel{d.}{=} \mathsf{U}[d:]$. The first statement in (b) follows from the facts that $\min\{d, D\}$ is determined by $\mathsf{U}[0 : d-1]$ and $\mathsf{refresh}_d(D)$ is determined by $\mathsf{U}[d:]$. The generalization in (b) similarly follows: the indicated random variables are functions of disjoint subsets of the random variables in U. (c) is proved as follows.

$$\begin{aligned} D &\leq \min\{i \geq d : \mathsf{U}[i] \leq \mathsf{FailureRate}[i]\} \\ &= d + \min\{i \geq 0 : \mathsf{U}[i+d] \leq \mathsf{FailureRate}[i+d]\} \\ &\leq d + \min\{i \geq 0 : \mathsf{U}[i+d] \leq \mathsf{FailureRate}[i]\} \\ &= d + \mathsf{refresh}_d(D). \end{aligned}$$

The non-decreasing failure rate condition is used in the second inequality. □

2.8.2 Defining CharString

In this section, we define CharString in the i.i.d. leader model. Without loss of generality, we assume all message delays have independent internal representations. The definition of CharString is the same as in the one-time leader model except for the way in which special honest slots are defined in positive time. In particular, the variables R_j are defined differently. For each $j \in \mathbb{N}$ such that $\mathsf{LeaderString}[T_j] = 0$, let

$$R_j = \mathsf{refresh}_{T_{j-1} - T_{j^*}}(\mathsf{delay}(T_{j^*} \to h_j)).$$

Just as before, T_j is a special honest slot (i.e., let $\mathsf{CharString}[T_j]$ be 0) if and only if $\mathsf{LeaderString}[T_j] = 0$ and $R_j < T_j - T_{j-1}$. When $T_{j^*} = T_{j-1}$ (i.e., the latest nonempty slot was special honest), $R_j = \mathsf{delay}(T_{j^*} \to h_j)$. Recall that this is precisely the definition of R_j in the one-time leader model. This modified definition of R_j allows us to prove Lemma 2.9 even if the sequence of R_js is not i.i.d. (see the proof given below).

Next, we show that the property of increasing heights of special honest blocks holds for the i.i.d. leader model as well. Note that h_j receives the message from the previous special honest slot at

time $T_{j^*} + \mathsf{delay}(T_{j^*} \to h_j)$. If T_j is special honest, then $R_j < T_j - T_{j-1}$. By Lemma 2.16(c), $\mathsf{delay}(T_{j^*} \to h_j) \leq (T_{j-1} - T_{j^*}) + R_j$. Put together, we get

$$T_{j^*} + \mathsf{delay}(T_{j^*} \to h_j) \leq T_{j^*} + (T_{j-1} - T_{j^*}) + R_j < T_{j-1} + T_j - T_{j-1} = T_j.$$

Thus, just as for the one-time leader model, the condition for T_j to be a special honest slot is sufficient, but not necessary, for h_j to have received the message from the previous special honest slot.

The final step is to show that the distribution of the random process $\mathsf{CharString}$ for the i.i.d. leader model is the same as it is for the one time leader model. In other words, we prove that Lemma 2.9 holds in the i.i.d. leader case as well.

Proof. It suffices to show the second claim for $j \geq 1$; all other statements follow from identical arguments as in the one-time leader case. Recall that $\mathsf{CharString}[T_j] = 0$ if and only if $\mathsf{LeaderString}[T_j] = 0$ and $R_j < T_j - T_{j-1}$. Consider the conditional probability distribution of the sequence $(R_j : j \in \mathbb{N}, \mathsf{LeaderString}[T_j] = 0)$ given $\mathsf{LeaderString}$. Statements (a) and (b) of Lemma 2.16 imply that, given $\mathsf{LeaderString}$,

- each R_j has the same distribution as Δ and
- the random variables $\tilde{R}_j = \min\{R_j, T_j - T_{j-1}\}$ are conditionally mutually independent.

Next, note that $R_j < T_j - T_{j-1}$ if and only if $\tilde{R}_j < T_j - T_{j-1}$. Finally, the sequence $(\mathsf{LeaderString}[T_j] : j \in \mathbb{N})$ is a Bernoulli sequence.

From these observations, the conditional independence of $(\mathsf{CharString}[T_j] : j \geq 1)$ follows, and so does the equation in the lemma. Thus, we have shown the second claim holds for $j \geq 1$. □

Chapter 3

Finality Gadgets and Blockchains with Dual Ledgers

3.1 Introduction

The longest-chain protocol, introduced by Nakamoto in Bitcoin [1], is the prototypical example of a blockchain protocol that operates in a permissionless setting. Put differently, the longest-chain protocol is *adaptive*: it remains safe and live irrespective of the number of active participants (nodes) in the system, as long as the fraction of adversarial nodes among the active ones is less than a half. Adaptivity (also known as dynamic availability) is a desirable property in a highly decentralized system such as a cryptocurrency. The downside of this protocol is that they are insecure during prolonged periods of network partition (asynchrony). This is unavoidable; miners in disconnected portions of the network keep extending their blockchains separately, unaware of the other chains. When synchrony resumes, one of the chains wins, which implies that blocks on the other chains get unconfirmed. These features are present not just in the longest-chain protocol, but also in other protocols that are derived from it, e.g., Prism [14].

In stark contrast, committee-based consensus protocols like PBFT [12] and Hotstuff [33] offer strong *finality* guarantees. These protocols remain safe even during periods of asynchrony, and regain liveness when synchrony resumes. Finality is also a desirable property for blockchains, as they may operate under conditions where synchrony cannot be guaranteed at all times. However, all finality-guaranteeing protocols come with a caveat: they are built for the permissioned setting. These protocols make progress only when enough number of votes have been accrued for each block. If a significant fraction of nodes become inactive (go offline), the protocol stalls completely. This prevents them from being adaptive.

We posit that the two disparate class of protocols is a consequence of the CAP theorem, a famous impossibility result in distributed systems. The theorem states that in the presence of a *network partition*, a distributed system cannot guarantee both *consistency* (safety) and *availability* (liveness) [34, 35]. The theorem has led to a classification of system designs, on the basis of whether they favor liveness or safety during network partitions. Blockchains, being distributed systems, inherit the trade-offs implicated by the CAP theorem. In particular, we see that the longest-chain class of protocols favor liveness while the committee-based protocols favor safety.

Recently, Lewis-Pye and Roughgarden [13] prove a CAP theorem for blockchains that highlights the adaptivity-finality trade-off explicitly. They show that "a fundamental dichotomy holds between

protocols (such as Bitcoin) that are adaptive, in the sense that they can function given unpredictable levels of participation, and protocols (such as Algorand) that have certain finality properties." The essence of this impossibility result is the following: it is difficult to distinguish network asynchrony from a reduced number of participants in the blockchain system. Therefore, a protocol is bound to behave similarly under both these conditions. In particular, adaptive protocols must continue to extend blockchains during asynchrony (thereby compromising finality), while finality-guaranteeing protocols must stall under reduced participation (which means they cannot have adaptivity).

In this chapter, we investigate whether the aforementioned trade-off between adaptivity and finality can be resolved at a user level, rather than at a system-wide level. In particular, we seek to build a blockchain system wherein all (honest) nodes follow a *common* block proposing mechanism, but different nodes can choose between two *different confirmation rules*. Under appropriate bounds on adversarial participation, one rule must guarantee adaptivity, while the other must guarantee finality. When the system is operating under desirable conditions, i.e., a large enough fraction of nodes are active and the network is synchronous, the blocks confirmed by both rules must coincide. Protocols with such a dual-confirmation rule (aka dual-ledger) design are termed as "Ebb-and-flow" protocols [27]. Such a design would be of interest in blockchains for many reasons. For example, for low-value transactions such as buying a coffee, a node may prefer liveness over safety, whereas for high-value transactions, it is natural to choose safety over liveness. Moreover, such a trade-off allows each node to make their own assumptions about the state of the network and choose a confirmation rule appropriately.

At a high level, we are inspired in our formulation from analogous designs to adapt the CAP theorem in practical distributed system settings (see Section 4 of the review by Gilbert and Lynch [35]). More concretely, dual-ledger designs do not fall under the purview of the CAP theorem [13]; the formulation assumes a single confirmation rule. A second motivation is that a variety of *finality gadgets* (in combination with a blockchain protocol) may also be viewed as providing a user-dependent dual ledger option. We elaborate on some recent proposals [36, 37, 38] in Section 3.2.

3.1.1 Our contributions

Our main contribution is to propose a new protocol, called the *checkpointed longest chain* protocol, which offers each node in the same blockchain system a choice between two different confirmation rules that have different adaptivity-finality trade-offs.

- **Block Proposal**. Just as in the longest chain protocol, honest miners build new blocks by extending the chain that they currently hold. In addition, some honest users participate in a separate *checkpointing protocol*, that marks certain blocks as checkpoints at regular intervals. An honest user adopts the *longest chain that contains the latest checkpoint*.

- **Confirmation rules**. The protocol's *adaptivity-guaranteeing* confirmation rule is simply the k-deep rule. An honest user confirms a block if it sees k blocks below it, for an appropriate choice of k. The protocol's *finality-guaranteeing* rule is for honest users to treat all blocks in its chain up to the last checkpointed block as confirmed.

The protocol is designed such that new blocks continue to be mined at increasing heights even if the participation level is low, but new checkpoints appear only if there is sufficient participation. This illustrates the adaptivity-guaranteeing property of the k-deep rule. On the other hand, checkpoints are guaranteed to be on a single chain irrespective of network conditions, while the longest chain containing the latest checkpoint may keep alternating between divergent chains. This implies the finality property of the checkpointing rule.

Below, we highlight the key design principles of our protocol, followed by the security guarantees.

Validity of Blockchains

Our protocol keeps intact the intrinsic *validity* of blockchains. Roughly, validity means that the set of blocks that are confirmed all lie on a single, monotonically increasing chain. Thus, the validity of a transaction can be inferred from just the blockchain leading up to the particular block. More precisely, an honest user that is constructing the block has the assurance that if the block is confirmed, all transactions in the block are confirmed, which means it accrues all the transaction rewards. Moreover, a user can infer that a certain transaction is confirmed simply by knowing that the transaction was included in a particular block and that the block was confirmed (as per either rule), without knowledge of the contents of other blocks. These properties are recognized to be important for the blockchain to be incentive-compatible and to enable light clients respectively. Further, validity of blockchains offers spam-resistance and is compatible with existing sharding designs. Although validity of blockchains is a common feature of a majority of protocols, recent high performance designs crucially decouple validation from consensus [14, 39]. The Snap-and-Chat design for the same problem [27] also does not have this feature.

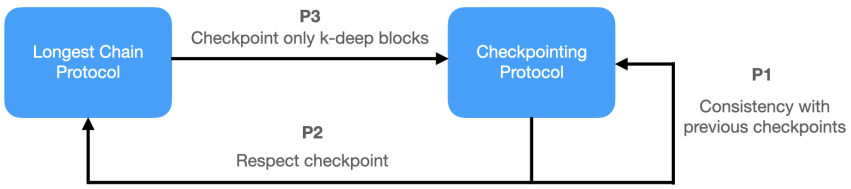

Figure 3.1: The Checkpointed Longest Chain

Interaction between Checkpointing and Longest Chain Protocol

In our design, the longest chain rule (i.e., the block proposal rule) and the checkpointing rule satisfy some constraints with respect to each other. The interlocking structure of constraints between the checkpointing protocol and the longest chain protocol is illustrated in Figure 3.1 and detailed below.

- **P1: Consistency with previous checkpoints.** The sequence of checkpoints must be on a chain. This places a self-consistency condition on the checkpointing protocol.
- **P2: Checkpointed longest chain rule.** The longest chain protocol respect the previous checkpoints. Honest users build adopt the longest chain that contains the latest checkpoint and mine new blocks at the tip of this chain.
- **P3: Checkpoints are deep enough.** The checkpointing protocol should only checkpoint blocks that are sufficiently deep in some honestly held chain. Furthermore, such a chain should be made available along with the checkpoint message.

All these three conditions are required to achieve our goals. Condition **P1** is a requisite for a validity preserving protocol (in particular, a validity preserving, finality-guaranteeing confirmation rule). The condition **P2** says that honest nodes should adopt the longest checkpointed chain, rather than the longest chain. Given **P1**, the condition **P2** is required to ensure new checkpoints are produced after a period of asynchrony. Without this condition **P2**, the block proposal rule is simply the longest chain rule. During asynchrony, the longest chain may be one that does not include some of the checkpoints, and there is no correction mechanism to ever bring the longest chain downstream of the last checkpoint. Condition **P3** ensures that during synchrony, any block that is eventually checkpointed would be a part of all honestly held chains. Thus, the dynamics of the protocol would be as if nodes are simply following the longest chain rule, and the known security guarantees would apply. If no such condition is placed, the checkpointing protocol could checkpoint an arbitrary block that possibly forks from the main chain by a large margin. The rule **P2** would force honest nodes to switch to this fork, thereby violating safety of the k-deep rule.

The Checkpointing Protocol

The checkpointing protocol in our design is a BFT consensus protocol. It is a slight modification of Algorand BA [40]. The protocol is extended from a single-iteration byzantine agreement protocol to a multi-iteration checkpointing protocol. In each iteration, nodes run the checkpointing protocol to checkpoint a new block, and **P1** guarantees that the sequence of checkpoints lie on a single chain. Coupled with the chain adoption rule (longest checkpointed chain), these checkpoints offer deterministic finality and safety against network partitions.

We state the safety guarantee of the checkpointing protocol as a basic checkpointing property (CP) as below:

- **CP0: Safety.** All honest users checkpoint the same block in one iteration of the checkpointing protocol, even during network partition. This checkpoint lies on the same chain as all previous checkpoints.

While one might consider different protocols for the checkpointing mechanism, the protocol must further satisfy some key properties during periods of synchrony.

- **CP1: Recency condition.** If a new block is checkpointed at some time, it must have been in a chain held by an honest user at some point in the *recent past*.
- **CP2: Gap in checkpoints.** The interval between successive checkpoints must be large enough to allow **CP1** to hold.
- **CP3: Conditional liveness.** If all honest nodes hold chains that have a common prefix (all but the last few blocks are common), then a new checkpoint will appear within a certain bounded time (i.e., there is an upper bound on the interval between two successive checkpoints).

CP1 helps in bounding the extent to which an honest user may have to drop honest blocks in its chain when it adopts a new checkpoint. Without this condition, (i.e., if arbitrarily old checkpoints can be issued), honest users may have to let go of many honestly mined blocks. This causes a loss in mining power, which we refer to as *bleeding*. Our choice of Algorand BA was made because it has this property. Other natural candidates such as PBFT [12] and HotStuff [33], do not have this property. We discuss this in detail in Section 3.9. **CP2** ensures that the loss in honest mining power due to the above mechanism is limited. This condition can be incorporated by design. **CP3** is a liveness property of the checkpointing protocol. It ensures new checkpoints appear at frequent intervals, leading to liveness of the finality-guaranteeing confirmation rule.

Security Guarantees (Informal)

We show the following guarantees for proof-of-work longest chain along with Algorand BA (augmented with the appropriate validity condition).

1. The k-deep rule is safe and live if the network is synchronous from the beginning and the fraction of adversaries among online users is less than a half.
2. The checkpointing protocol, which is safe by design, is also live soon after the network partition is healed under the partially synchronous model.
3. The ledger from the checkpoint rule is a prefix of the ledger from the k-deep rule from the point of view of any user.

Proof Technique

We prove these security guarantees using the following strategy. First, we show that if all honest users hold chains that obey the k-common prefix condition for an extended period of time, then new blocks get checkpointed at regular intervals, and these blocks are part of the common prefix of the honestly held chains. This tells us that under conditions in which the vanilla longest chain rule is secure, the checkpointed longest chain rule has exactly the same dynamics. Therefore, it inherits the same security properties of the longest chain rule. Secondly, we show that once sufficient time has passed after a network partition is healed, the chains held by the honest user under the checkpointed longest chain rule are guaranteed to have the common prefix property. Coupled with the first result, we infer that new checkpoints will eventually be confirmed thereby proving liveness of the checkpoint-based confirmation rule under partial synchrony.

Outline

We review related works in Section 3.2 and place our results in the context of several recent works on the same topic as this chapter. The similarities and differences in the techniques in our work with closely related works is pictorially illustrated in Figure 3.2. We state our network and security models formally in Section 3.3. In Section 3.4, we describe the checkpointed longest chain protocol, highlighting the roles of the miners and precisely stating the two confirmation rules. For clarity, we describe the checkpointing protocol as a black box with certain properties here; we give the full protocol in Section 3.6. We state our main theorem, concerning the security guarantees of our protocol, in Section 3.5. In the rest of the section, we give a proof sketch of the theorem. The formal proofs are given in Appendices 3.7 and 3.8. We conclude the chapter with a discussion and pointers for future work in Section 3.10.

3.2 Related Work

CAP Theorem The formal connection between the CAP theorem and blockchains was recently made by Lewis-Pye and Roughgarden [13]. They provide an abstract framework in which a wide class of blockchain protocols can be placed, including longest chain protocols (both Proof-of-Work based [4] and PoS based [41]) as well as BFT-style protocols [42]. The main result of Lewis-Pye and Roughgarden [13] says that a protocol (containing a block "encoding" procedure and a block confirmation rule) which is adaptive (i.e., which remains live in an unsized setting) cannot offer finality (i.e., deterministic safety, even under arbitrary network conditions) and vice-versa. However, the same chapter is mute on the topic of whether different block confirmation rules could offer different guarantees, which is the entire focus of this work. We are inspired in our formulation from analogous designs to adapt the CAP theorem in practical distributed system settings (see Section 4 of Gilbert and Lynch [35]).

User-Dependent Confirmation Rules The idea of giving users the option of choosing their own confirmation rule, based on their beliefs about the network conditions and desired security level was pioneered by Nakamoto themself, via choosing the value of k in the k-deep confirmation rule. A recent work [43] allows users the option of choosing between partially synchronous confirmation rule and a synchronous one. However, *neither* of the confirmation rules are adaptive, as the block proposal rule itself is a committee-based one and cannot make progress once the participation is below a required level. Therefore, they do not "break" the CAP theorem as proposed by Lewis-Pye and Roughgarden [13]. In contrast, our protocol offers users the choice of an adaptive system.

Hybrid Consensus Hybrid consensus [44] is a different technique of incorporating a BFT protocol into the longest-chain protocol. In a nutshell, the idea in this design is to use the longest-chain protocol to randomly select a committee, which then executes a BFT protocol to confirm blocks. The committee consists of the miners of the blocks on the longest chain over a shifting interval of constant size, and is thus re-elected periodically. Hybrid consensus addresses the problem of building a responsive protocol (i.e., latency proportional to network delay) in a PoW setting. It does not, however, address the adaptivity-finality dilemma. In fact, as we illustrate here, the protocol offers neither adaptivity nor finality. Once a miner is elected as a committee member, it is obliged to stay active until the period of its committee is over, which compromises on the adaptivity property. As for finality, although the blocks are confirmed by a finality-based protocol, the committee election mechanism compromises the finality guarantee. During asynchrony, nodes cannot achieve consensus on who belongs to the committee; thus, the whole protocol loses safety.

Checkpointing Checkpointing is a protocol run on top of the longest chain protocol (run by largely honest parties) that deterministically marks certain blocks as finalized; a simple form of checkpointing was pioneered, and maintained until 2014, by Satoshi Nakamoto (presumably honest). In the context of our work, checkpointing provides finality (essentially by fiat) while the longest chain rule ensures adaptivity. We can interpret our chapter as an attempt to provide appropriate conditions on the interaction between the checkpointing and the longest chain protocol, and also to show how to realize such checkpointing in a distributed manner. We note that [38] is a recent work that studies the checkpointed ledger when the longest-chain protocol has supermajority adversaries; this regime is different from that studied in the present chapter, where we assume that the longest-chain protocol has majority honest and the checkpointing protocol has $2/3$ honest users. For completeness, we have included the architecture of checkpointed ledger in the comparison in Figure 3.2.

Casper FFG and Gasper Casper FFG (*Casper: the Friendly Finality Gadget*) [45] pioneered the study of finality gadgets. Casper FFG also introduced the notion of "economic security". The finality gadget allows cryptographic proofs of malicious behavior which can be used to disincentivize such behavior. We do not explore this aspect of finality gadgets in our chapter. Casper FFG is

not a completely specified protocol [45]. A recent follow-up chapter called Gasper [46] provides a complete protocol. Gasper can be viewed as a checkpointed longest chain protocol, with the longest chain rule replaced by the *Latest Message Driven Greediest Heaviest Observed Subtree* (LMD GHOST) rule and the checkpointing protocol being Casper FFG. Gasper does not provide formal security guarantees; in fact, the work by Neu et al. [27] shows a liveness attack on Gasper.

Afgjort Afgjort [36] is a recent finality gadget proposed by Dinsdale et al., which formally describes the desirable properties that a finality gadget must have, some of them are similar to the ones we require as well. Afgjort has a two-layer design: it takes an off-the-shelf longest chain protocol and adds a finality layer to it. Such a system can work as desired when the network is synchronous, including when the participation levels are variable. However, it is destined to fail in one of two ways in a partially synchronous setting: either the finality gadget stops finalizing new blocks (i.e., violates liveness), or it finalizes "conflicting" blocks, i.e., blocks in two different chains (i.e., violates validity). Thus Afgjort does not provide the guarantees we seek in this work. More generally, any layer-two finality gadget on top of an adaptive protocol would not meet the desired objectives for the same reason; they would be missing rule **P2** (see Figure 3.2).

GRANDPA GRANDPA [37] is another recent finality gadget, proposed by Stewart et al. Just as in our protocol (and unlike Afgjort), GRANDPA alters the underlying block production mechanism to respect blocks that has been finalized by the finality gadget. GRANDPA's security relies on the assumption that the block-production mechanism (longest-chain rule/GHOST) provides a type of "conditional eventual consensus". I.e., If nodes keep building on the last finalized block and don't finalize any new blocks, then eventually they have consensus on a longer chain. This argument, though presumably true, is not a rigorous security analysis; in our work, we prove security for a completely specified model and protocol.

More importantly, comparing Figures 3.1 and 3.2, we see that GRANDPA does not have property **P3**. This leads to a security vulnerability in the variable participation setting. The issue arises due to the following subtlety: in the variable participation setting, a particular block could be 'locked' onto by the checkpointing protocol, but could be checkpointed much later (this does not happen under full participation). A block at the tip of the longest chain could soon be displaced from it by the adversary. If this block is locked while it is on the chain, but checkpointed only when it is out of the chain, the safety of the k-deep rule will be violated. A more detailed description of this attack is given in Section 3.9.

Snap-and-Chat protocols A concurrent work by Neu et al. [27] also solves the adaptivity vs finality dilemma posed by the CAP theorem. (Their work appeared online a few weeks prior to ours). They introduce a technique, called Snap-and-Chat, to combine a longest chain protocol with any BFT-style one to obtain a protocol that has the properties we desire. More specifically, their protocol produces two different ledgers (ordered list of transactions), one that offers adaptivity and

the other that offers finality. A user can choose to pick either one, depending on its belief about the network condition. These ledgers are proven to be consistent with each other (the finality guaranteeing ledger is a prefix of the adaptive one).

While the outcomes of the Snap-and-Chat design are analogous to ours, the approach adopted and the resulting blockchain protocol are quite different. A major advantage of this approach is its generality: unlike other existing approaches, it offers a blackbox construction and proof which can be used to combine a variety of adaptive and finality preserving protocols. However, the Snap-and-Chat design has an important caveat: the sequence of blocks that are confirmed in their protocol do not necessarily form a single chain. Indeed, in the partially synchronous setting, the finality-guaranteeing confirmation rule can finalize blocks on two different forks one after the other. The Snap-and-Chat protocol overcomes this apparent issue by constructing a ledger after the blocks have been finalized, and "sanitizing" it at a later step.

The sanitization step implies that not all transactions in a confirmed block may be part of the ledger: for example, if it is a double spend relative to a transaction occurring earlier in the ledger, such transaction will be removed. Put differently, the validity of a particular transaction in a particular block is decided only once the block is confirmed, and not when the block is proposed. Most practical blockchain systems are designed with coupled validation, i.e., an honest block proposer can ensure that blocks contain only valid transactions. Our approach maintains coupled validity of blockchains and does not have this shortcoming, as highlighted in Section 3.1.

Many of these related works can be placed in a common framework, as illustrated in Figure 3.2. In Ebb-and-Flow [27], previous checkpoints are not respected by either the longest chain protocol or the checkpointing protocol. Hence checkpoints may not form a single chain and the final ledger is constructed by sanitization, which makes it impossible to have validation before block proposal. Afgjort [36] takes an off-the-shelf longest chain protocol and keeps checkpointing blocks on it. Due to lack of interaction between the longest chain protocol and the checkpointing protocol, the protocol may fail in a partially synchronous setting when the security of the blockchain is broken. The Checkpointed Ledger [38] and GRANDPA [37] are very similar to our design; the difference is that the checkpoints don't need to be buried deep enough in the longest chain. This minor change makes their protocols become insecure under the variable participation setting.

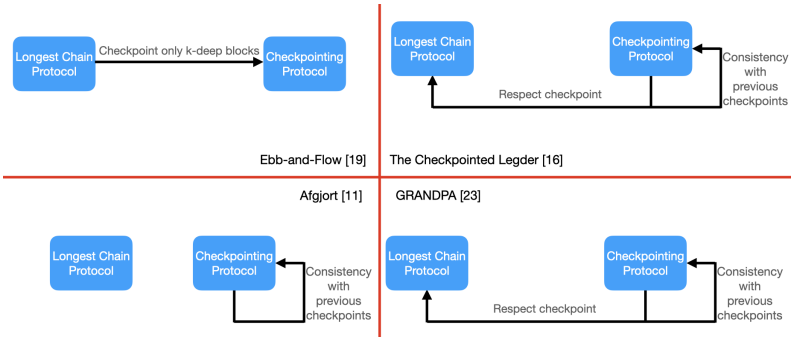

Figure 3.2: A comparison of related work

3.3 Security Model

Environment. A blockchain protocol Π is directed by an *environment* $\mathcal{Z}(1^\kappa)$, where κ is the security parameter, i.e., the length of the hash function output. This environment (i) initiates a set of participating nodes $\mathcal{N}$; (ii) manages a public-key infrastructure (PKI) and assigns each node with a unique cryptographic identity; (iii) manages nodes through an adversary $\mathcal{A}$ which *corrupts* a subset of nodes before the protocol execution starts; (iv) manages all accesses of each node from/to the environment including broadcasting and receiving messages.

Network model. The nodes' individual timers do not need to be synchronized or almost synchronized. We only require they have the same speed. In our network, we have a variety of messages, including blocks, votes, etc. The following message delay bounds apply to all messages. Secondly, all messages sent by honest nodes are broadcast messages. Thirdly, all honest nodes re-broadcast any message they have heard. The adversary does not suffer any message delay. In general, we assume that the adversary controls the order of delivery of messages, but the end-to-end network delay between any two honest nodes is subject to some further constraints that are specified below. We operate under either one of the two settings specified below:

- **M1 (Partial synchrony model):** A global stabilization time, GST, is chosen by the adversary, unknown to the honest nodes and also to the protocol designer. Before GST, the adversary can delay messages arbitrarily. After GST, all messages sent between honest nodes are delivered within Δ time. Moreover, messages sent by honest nodes before GST are also delivered by $\mathsf{GST} + \Delta$.
- **M2 (Synchrony model):** All messages sent from one honest node to another are delivered within Δ time.

Participation model. We introduce the notion of *sized/unsized* number of nodes to capture the notion that node network activity (online or offline) varies over time. It is important to note that we allow for adversarial nodes to go offline as well. A node can come online and go offline at any time during the protocol. An online honest node always executes the protocol faithfully. An offline node, be it honest or adversarial, does not send any messages. Messages that are scheduled to be delivered during the time when the node is offline are delivered immediately after the node comes online again. We consider two different scenarios:

- **U1 (Sized setting):** All nodes stay online at all times.
- **U2 (Unsized setting):** Nodes can come online and offline at arbitrary times, at the discretion of the adversary, provided a certain minimal number of nodes stay online.

Abstract protocol model. We describe here our protocol in the abstract framework that was developed by Lewis-Pye and Roughgarden [13]. In this framework, a blockchain protocol is specified as a tuple $\Pi = (I, O, C)$, where I denotes the *instruction set*, O is an *oracle* that abstracts out

the leader election process, and C is a confirmation rule (e.g., the k-deep confirmation rule). The bounds on adversarial ratios are also specified through O. The rationale for adopting this framework is to point out precisely how we circumvent the CAP theorem for blockchains [13].

The theorem states that no protocol $\Pi = (I, O, C)$ can simultaneously offer both finality (safety in the partially synchronous setting) and adaptivity (liveness in the unsized setting). Our checkpointed longest chain protocol is a 4-tuple $\Pi_{\text{CLC}} = (I, O, C_1, C_2)$, with two confirmation rules C_1 and C_2. This entire protocol can be split at a node level into two protocols: $\Pi_{\text{fin}} = (I, O, C_1)$ and $\Pi_{\text{ada}} = (I, O, C_2)$. The protocol Π_{fin} guarantees (deterministic) safety and (probabilistic) liveness under network assumption M1 and participation level U1, just as many BFT-type consensus protocols do. It therefore guarantees finality. Π_{ada} guarantees safety and liveness under network assumption $M2$ and participation level $U2$, just as longest chain protocols do. It thereby guarantees adaptivity. Finally, both Π_{fin} and Π_{ada} are safe and live under conditions $M2$ and $U1$; moreover, the set of blocks confirmed by Π_{fin} at any time is a subset of those confirmed by Π_{ada}. A similar property is proven by Neu et al. [27].

Each node can choose one of the two confirmation rules according to their demands and assumptions. We specify the exact specifications of I, O, C_1 and C_2 and the associated security properties in the next section.

3.4 Protocol Description

Nodes in the protocol. In our protocol, let $\mathcal{N}$ denote the set of all participating nodes. Among these, there are two subsets of participating nodes $\mathcal{N}_1$ and $\mathcal{N}_2$. We call nodes in $\mathcal{N}_1$ *miners*, whose role is to propose new blocks. We call nodes in $\mathcal{N}_2$ *checkpointers*, whose role is to vote for blocks on the blockchains they currently hold and mark them as *checkpoints*. Note that we allow for any relation among these sets, including the possibility that all nodes play both roles ($\mathcal{N}_1 = \mathcal{N}_2 = \mathcal{N}$), or the two kinds of nodes are disjoint ($\mathcal{N}_1 \cap \mathcal{N}_2 = \phi$). The instruction set I and the oracle O is different for miners and checkpointers, and is specified below. Note that in the sized/unsized setting, both $\mathcal{N}_1$ and $\mathcal{N}_2$ are sized/unsized.

Before we specify the protocol, we introduce some terminology pertaining to checkpoints. A *checkpoint certificate* is a set of votes from at least 2/3 of the checkpointers for a certain block, that certifies that the block is a checkpoint. When an honest node receives both a checkpoint certificate and a chain that contains the block being checkpointed, we say that the honest node has heard of a checkpoint. We say that a checkpoint *appears at time* t if the t is the first time that an honest node hears of it.

I for all nodes. Every honest node holds a single blockchain at all times, which may be updated upon receiving new messages. They all follow the *checkpointed longest chain rule*, which states that a node selects the longest chain *that extends the last checkpointed block* it has heard of so far. Ties are broken by the adversary. To elaborate, a node keeps track of the last checkpoint it has heard

of so far. If a node receives a longer chain that includes the last checkpoint, it adopts the received new chain. A node ignores all chains which do not contain the last checkpoint block as per their knowledge.

I, O **for miners.** An honest miner adopts the tip of its checkpointed longest chain as the parent block. The miner forms a new block with a *digest* of the parent block and all transactions in its buffer, and broadcasts it when it is chosen as a leader by the oracle. The oracle O chooses miners as leaders at random intervals, that can be modeled as a Poisson process with rate λ. This joint leader election process can be further split into two independent Poisson processes. Let the fraction of online adversarial miners be $\beta < 1/2$. The honest blocks arrive as a Poisson process of rate $(1-\beta)\lambda$, while the adversarial blocks arrive as an independent Poisson process of rate $\beta\lambda$.

I, O **for checkpointers.** The checkpointers run a multi-iteration Byzantine Agreement (BA) protocol Π_{BA} to checkpoint blocks, with each iteration checkpointing one block. Our protocol is a slight variant of the Algorand BA protocol [40]. The complete protocol is described in Section 3.6 with a minor modification from Algorand, which is highlighted in red. This modification is made so as to enable a consistency check across iterations without losing liveness (i.e., to get a multi-iteration BA from a single iteration one).

Briefly, the protocol works as follows. Each iteration is split into *periods*. Each period has a unique leader chosen by the oracle. Nominally, the values on which consensus is to be achieved amongst all checkpointers are the blockchains held by the checkpointers. In practice, the values may be the hashes of the last block of the blockchain. A key difference from the Algorand protocol is that we allow the checkpointers to change their values at the beginning of each period. The checkpointers aim to achieve consensus amongst these values. The final chain that has been agreed upon is broadcast to all honest nodes (not just checkpointers) together with a certificate. The block that is exactly k-deep in this chain is chosen as the checkpoint block in current iteration. Here, k is a parameter of the protocol Π_{CLC}, and is chosen to be $\Theta(\kappa)$. We call it the *checkpoint depth parameter*.

Confirmation rules

We propose the following two confirmation rules for Π_{CLC}, which have different security guarantees under different assumptions. Either rule can be adopted by any node in the protocol

- C_1: Confirm the chain up to the last known checkpoint.
- C_2: Confirm all but the last k' blocks in the checkpointed longest chain, i.e., in the chain that is currently held, where $k' = \Theta(\kappa)$

Note that every honest node may choose any value of k' that they wish.

Intuition behind the checkpointing protocol

The checkpointing protocol is designed such that under optimistic conditions, a checkpoint is achieved within one period. These optimistic conditions are that the leader of a period is honest, the network is synchronous, and all chains held by honest checkpointers satisfy the common prefix property. The sequence of leaders for this protocol is guaranteed to be chosen in an i.i.d. fashion amongst online checkpointers by the oracle O. We assume that the fraction of adversarial checkpointers is $\beta' < 1/3$. Thus, the probability of selecting an adversarial leader at any round is $< 1/3$.

We now state some key properties of the checkpointing protocol. These properties, **CP0, CP1, CP2, CP3** were introduced in Section 3.1 and are elaborated upon below.

- **CP0: Safety.** All honest nodes checkpoint the same block in one iteration of the checkpointing protocol, even during network partition. This checkpoint lies on the same chain as all previous checkpoints. This is a safety property of the checkpointing protocol, which will be essential in guaranteeing the safety of Π_{fin}.

- **CP1: Recency condition.** If a new block is checkpointed at some time t by an honest node for the first time, it must have been in a chain held by an honest node at some time $t' \geq t - d$. Here, d is the *recency* parameter of the protocol, and is of the order $O(\sqrt{\kappa}\Delta)$.

- **CP2: Gap in checkpoints.** If a new block is checkpointed at some time t by an honest node for the first time, then the next iteration of the checkpointing protocol (to decide the next checkpoint) will begin at time $t+e$. Here, e is the *inter-checkpoint interval* and is chosen such that $e \gg d$.

- **CP3: Conditional liveness.** If all honest nodes hold chains that are self-consistent (they satisfy k-common prefix property), during an iteration of the checkpointing protocol, then the checkpointing protocol finishes within $O(\Delta)$ time. Thus, in a period where all honest nodes hold chains that are self-consistent, checkpoints appear within $e + O(\Delta)$ time of each other.

In Section 3.7, we prove that the checkpointing protocol we use (modified Algorand BA) satisfies these properties. As such, any protocol that satisfies **CP0 - CP3** can be used in our design.

3.5 Main Result

To state the main security result of our protocol, recall the notations set in Section 3.3. The complete protocol is denoted by $\Pi_{\text{CLC}} = (I, O, C_1, C_2)$ and its two user-dependent variations are $\Pi_{\text{fin}} = (I, O, C_1)$ and $\Pi_{\text{ada}} = (I, O, C_2)$. We also recall the notation $\text{M1}, \text{M2}$ for the partially synchronous and synchronous network model, and U1, U2 for the sized and unsized participation

models. Clearly, M1 is a more general setting than M2 and U2 is a more general setting than U1. In Theorem 3.1, a result stated for a general setting also applies to the more restricted setting, but not vice-versa. We first define the notion of safety and liveness that we use for our protocols.

Definition 3.1 (Safety)**.** *A blockchain protocol* Π *is safe if the set of blocks confirmed during the execution is a non-decreasing set. Put differently,* Π *is safe if a block once confirmed by the confirmation rule remains confirmed for all time thereafter.*

Definition 3.2 (Liveness)**.** *A blockchain protocol* Π *is live if there exists constants* $c, c' > 0$ *s.t. the number of new honest blocks confirmed in any interval* $[r, s]$ *is at least* $\lfloor c(s-r) - c' \rfloor$*.*

We now state our main theorem, concerning the safety and liveness of Π_{fin} and Π_{ada}.

Theorem 3.1. *Assume the fraction of adversarial mining power among total honest mining power is bounded by* $\beta < 1/2$*. Further, assume the fraction of adversarial checkpointers is always less than* $1/3$*. Then, the protocol* Π_{CLC} *has the following security guarantees for an execution that runs for a duration of* $T_{\max} = O(\text{poly}(\kappa))$ *time, where* κ *is the security parameter, :*

- ***Security of*** Π_{fin}***:*** *The protocol* Π_{fin} *is safe, and is live after* $O(\mathsf{GST} + \kappa)$ *time in the setting (*$\text{M1}, \text{U1}$*), except with probability negligible in* κ*.*
- ***Security of*** Π_{ada}***:*** *The protocol* Π_{ada} *is safe and live in the setting (*$\text{M2}, \text{U2}$*), except with probability negligible in* κ*.*
- ***Nested Protocols:*** *At any time* t*, the set of blocks confirmed by* Π_{ada} *is a superset of the set of blocks confirmed by* Π_{fin} *in all settings.*

We provide a proof sketch for Theorem 3.1 below. A detailed proof is relegated to Appendices 3.7 and 3.8.

Proof sketch

Our proof for Theorem 3.1 can be split into two parts. First, Section 3.7, we show that our checkpointing protocol Π_{BA} satisfies the four checkpointing properties given in Section 3.4, namely **CP0, CP1, CP2, CP3**. Then, in Section 3.8, we show that our complete protocol Π_{CLC} satisfies the desired security properties if it uses *any* sub-protocol for checkpointing that satisfies the above properties. We first highlight some aspects of Theorem 3.1 that are straightforward to deduce, assuming that Π_{BA} does satisfy the aforementioned properties.

Firstly, the safety of Π_{fin} under setting $(\text{M1}, \text{U1})$ can be deduced from two facts: 1) (**CP0**) safety of Π_{BA} holds under setting $(\text{M1}, \text{U1})$, and 2) (**P1**) the checkpointed longest chain rule respects all previous checkpoints (see Fig. 3.3). Specially, **CP0** of Π_{BA} ensure that all checkpoints are uniquely decided by all honest nodes and they lie on a single chain by **P1**, while the checkpointed longest

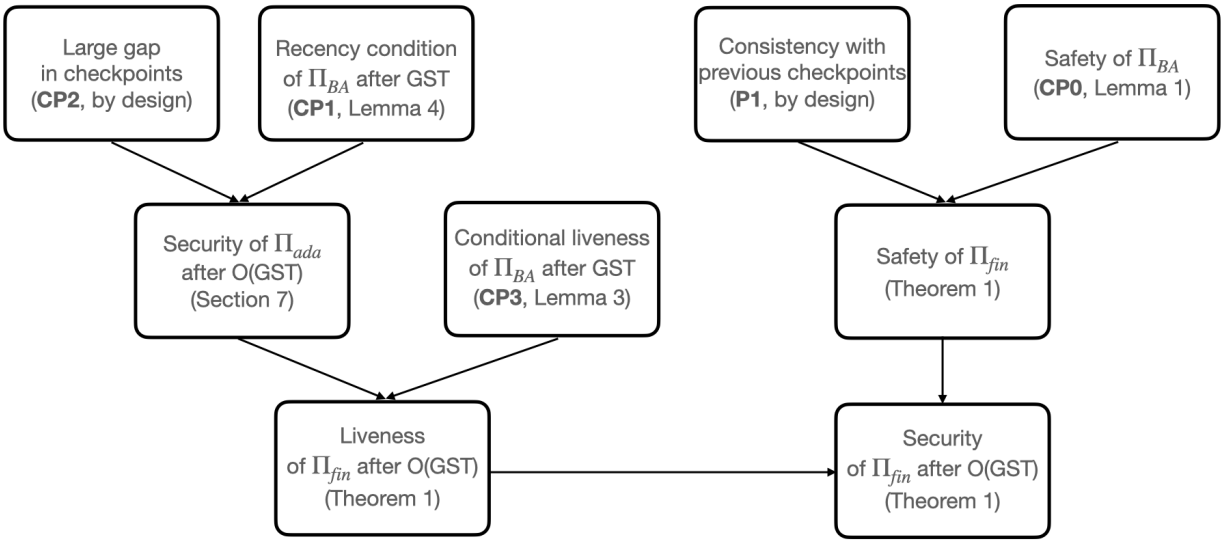

Figure 3.3: Flowchart for proving the security property of Π_{fin} under setting $(\text{M1}, \text{U1})$

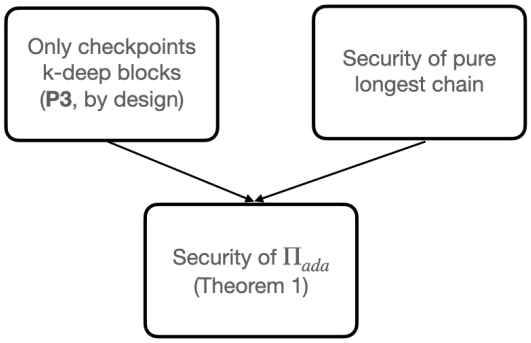

Figure 3.4: Flowchart for proving the security property of Π_{ada} under setting $(\text{M2}, \text{U2})$

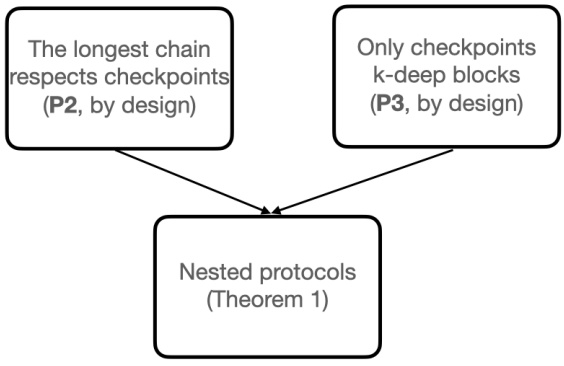

Figure 3.5: Flowchart for proving the nesting property of Π_{fin} and Π_{ada}

chain protocol instructs honest nodes to always adopt the longest chain with the last checkpoint block. Therefore, an honest node will always keep every checkpoint block it has seen forever.

Secondly, the security of Π_{ada} under setting $(\text{M2}, \text{U2})$ are deduced by two facts: 1) security of pure longest chain rule; 2) (**P3**) the checkpointing protocol respects the k-deep rule (see Fig. 3.4). **P3** ensures that every checkpoint must be at least k-deep in the chain of some honest node when it is decided by Π_{BA}, while a block that is k-deep should remain in the longest chain of all honest nodes forever with high probability under synchrony [6, 30, 7]. Therefore, every checkpoint will already be present in all honest nodes' chains when it appears. Thus, the chains held by the nodes of the protocol are indistinguishable from the case where they are following the pure longest chain protocol. Given that the confirmation rule is also the same for both protocols (k-deep rule), Π_{ada} inherits the safety and liveness properties of the pure longest chain protocol.

Thirdly, the nesting property of the confirmed blocks also follow immediately by design, given that any checkpointed block must have been already k-deep in a node's chain (**P3**) and the checkpointed longest chain is defined as the longest chain that contains the latest checkpoint (**P2**), (see Fig. 3.5).

It remains to prove the liveness guarantee of Π_{fin} under setting $(\text{M1}, \text{U1})$. In Section 3.7, we evaluate the safety and liveness of our checkpointing protocol Π_{BA} – a modified version of Algorand BA. We show that it satisfies the desired checkpointing properties listed in Section 3.1. A distinguishing property of Π_{BA} is the recency condition (**CP1**). This property states that Π_{BA} only outputs checkpoints that are recently contained in some honest node's chain. Without this property, the adversary could make all honest nodes waste a long time mining on a chain that will never be checkpointed.

In Section 3.8, we prove an important result of Π_{ada} under partial synchronous model: the safety and liveness property will hold after $O(\text{GST})$ time. This is essential to guarantee the liveness of Π_{fin} as a block can be only checkpointed when it lies in the k-common prefix of the checkpointed longest chains of all honest nodes. Finally, combining all these results (see Fig. 3.3), we show that Π_{fin} is live after $O(\text{GST})$ time with high probability, thereby completing the proof of the main theorem.

3.6 Algorand BA is a Checkpointing Protocol

We outline the full Algorand Byzantine Agreement (BA) protocol below for completeness. A minor modification (marked in red), adding validation, is the only addition to the original protocol. Honest checkpointers run a multi-iteration BA to commit checkpoints. The goal of the i-th iteration is to achieve consensus on the i-th checkpoint. Each iteration is divided into multiple periods and view changes will happen across periods.

All checkpointers start period 1 of iteration 1 at the same time (time 0). Checkpointer i starts period 1 of iteration n after it receives $2t + 1$ cert-votes for some value v for the same period p of

iteration $n-1$ and waits for another fixed time e, and only if it has not yet started an iteration $n' > n$. Checkpointer i starts period $p \geq 2$ of iteration n after it receives $2t+1$ next-votes for some value v for period $p-1$ of iteration n, and only if it has not yet started a period $p' > p$ for the same iteration n, or some iteration $n' > n$. For any iteration n, checkpointer i sets its starting value for period $p \geq 2$, st_i^p, to v (the value for which $2t+1$ next-votes were received and based on which the new period was started). For $p = 1$, $st_i^1 = \perp$. The moment checkpointer i starts period p of iteration n, he finishes all previous periods and iterations and resets a local timer $clock_i$ to 0. At the beginning of every period, every honest checkpointer i sets v_i to be the checkpointed longest chain in its view at that time.

Each period has a unique leader known to all checkpointers. The leader is assigned in an i.i.d. fashion by the permitter oracle O. Each checkpointer i keeps a timer $clock_i$ which it resets to 0 every time it starts a new period. As long as i remains in the same period, $clock_i$ keeps counting. Recall that we assume the checkpointers' individual timers have the same speed. In each period, an honest checkpointer executes the following instructions step by step.

Step 1: [Value Proposal by leader] The leader of the period does the following when $clock_i = 0$; the rest do nothing. If $(p = 1)$ OR $((p \geq 2)$ AND (the leader has received $2t+1$ next-votes for $\perp$ for period $p-1$ of iteration $n))$, then it proposes its value v_i. Else if $((p \geq 2)$ AND (the leader has received $2t+1$ next-votes for some value $v \neq \perp$ for period $p-1$ of iteration $n))$, then the leader proposes v.

Step 2: [The Filtering Step] Checkpointer i does the following when $clock_i = 2\Delta$. If$(p = 1)$ OR $((p \geq 2)$ AND (i has received $2t+1$ next-votes for $\perp$ for period $p-1$ of iteration $n))$, then i soft-votes the value v proposed by the leader of current period if (it hears of it) AND (the value is VALID OR i has received $2t+1$ next-votes for v for period $p-1$). Else if $((p \geq 2)$ AND (i has received $2t+1$ next-votes for some value $v \neq \perp$ for period $p-1$ of iteration $n))$, then i soft-votes v.

Step 3: [The Certifying Step] Checkpointer i does the following when $clock_i \in (2\Delta, 4\Delta)$. If i sees $2t+1$ soft-votes for some value $v \neq \perp$, then i cert-votes v.

Step 4: [The Period's First Finishing Step] Checkpointer i does the following when $clock_i = 4\Delta$. If i has certified some value v for period p, he next-votes v. Else if ($(p \geq 2)$ AND (i has seen $2t+1$ next-votes for $\perp$ for period $p-1$ of iteration $n))$, he next-votes $\perp$. Else he next-votes his starting value st_i^p.

Step 5: [The Period's Second Finishing Step] Checkpointer i does the following when $clock_i \in (4\Delta, \infty)$ until he is able to finish period p. If i sees $2t+1$ soft-votes for some value $v \neq \perp$ for period p, then i next-votes v. If $((p \geq 2)$ AND (i sees $2t+1$ next-votes for $\perp$ for period $p-1$) AND (i has not certified in period $p))$, then i next-votes $\perp$.

Halting condition: Checkpointer i HALTS current iteration if he sees $2t+1$ cert-votes for some value v for the same period p, and sets v to be his output. Those cert-votes form a certificate for v. The block that is exactly k-deep in v is chosen as the checkpoint in current iteration.

A proposed value v (with block B being exactly k-deep in it) from period p of iteration n is VALID for checkpointer i (in the same period and iteration) if:

- Value v is proposed by the leader of that period;
- Block B is a descendant of all previously checkpointed blocks with smaller iteration number;
- Block B is contained in the checkpointed longest chain that the checkpointer i holds when entering period p.

The only modification to the protocol is in Step 2, in the first condition, where the notion of validity is introduced. This is the only place where new proposals are considered. The validity notion helps transform the Algorand BA protocol into the multi-iteration checkpointing protocol that we desire. In Section 3.7, we show that this protocol indeed satisfies properties **CP0-CP3**, as mentioned in Section 3.4.

3.7 Checkpointing Properties

In this section, we prove several important properties of our modified Algorand BA protocol. We assume throughout that the number of adversarial checkpointers is strictly less than one-third the total number of checkpointers; this is necessary to achieve any security results for a partially synchronous consensus protocol such as Algorand BA.

- It is safe even under asynchronous network conditions(**CP0**, proved in Lemma 3.1). Moreover, it remains deadlock free, in the sense that it can always proceed as long as messages will be delivered eventually (Lemma 3.2).
- It is live after GST if the local checkpointed longest chains of all honest nodes satisfy common prefix property (**CP3**, proved in Lemma 3.3).
- It only outputs recently inputted values after GST (**CP1**, proved in Lemma 3.4).

We first define some useful notations according to the BA protocol described in Section 3.6.

Definition 3.3 (Potential starting value for period p)**.** *A value v that has been next-voted by $t+1$ honest nodes for period $p-1$.*

Definition 3.4 (Committed value for period p)**.** *A value v that has been cert-voted by $2t+1$ nodes for period p.*

Definition 3.5 (Potentially committed value for period p)**.** *A value v that has been cert-voted by $t+1$ honest nodes for period p.*

Although we slightly altered Algorand BA protocol (which is highlighted in red in Section 3.6), we note that our modification does not break the safety of the protocol or cause any deadlock in Lemma 3.1 and Lemma 3.2. At a high level, the validity check only causes less soft-votes from honest nodes, which is indistinguishable with the case where the leader is malicious and no value

receives at least $2t + 1$ soft-votes in some period. Therefore, the safety and deadlock-free property remain.

Lemma 3.1 (Asynchronous Safety, **CP0**)**.** *Even when the network is partitioned, the protocol ensures safety of the system so that no two honest nodes will finish one iteration of the protocol with different outputs.*

Proof. The following properties hold even during a network partition.

- By quorum intersection, as each honest node only soft-votes one value, then at most one value is committed or potentially committed for each period p in one iteration.
- If a value v is potentially committed for period p, then only v can receive $2t + 1$ next-votes for period p. Thus, the unique potential starting value for period $p + 1$ is v.
- If a period p has a unique potential starting value $v \neq \perp$, then only v can be committed for period p. Moreover, honest nodes will only next-vote v for period p, so the unique potential starting value for period $p + 1$ is also v. Inductively, any future periods $p' > p$ can only have v as a potential starting value. Thus, once a value is potentially committed, it becomes the unique value that can be committed or potentially committed for any future period, and no two honest nodes will finish this iteration of the protocol with different outputs.

□

Lemma 3.2 (Asynchronous Deadlock-freedom)**.** *As long as messages will be delivered eventually, an honest node can always leave period p, either by entering a higher period or meeting the halting condition for the current iteration.*

Proof. We first prove that there can never exist $2t + 1$ next-votes for two different non-$\perp$ values from the same period p by induction.

Start with $p = 1$. Note that every honest node sets $st_i^1 = \perp$ and at most one value (say v) could receive more than $2t + 1$ soft-votes. Therefore only value v and $\perp$ could potentially receive more than $2t + 1$ next-votes in period 1. Note that it is possible that both v and $\perp$ receive more than $2t + 1$ next-votes: all the honest nodes could next-vote for $\perp$ in Step 4 and then next-vote for v in Step 5 after seeing the $2t + 1$ soft-votes for v.

Assume that the claim holds for period $p - 1$ ($p \geq 2$): there exist at most two values each of which has $2t+1$ next-votes for period $p-1$, and one of them is necessarily $\perp$. Then there are three possible cases:

- Case 1: Only $\perp$ has $2t+1$ next-votes for period $p-1$. This is the same as the induction basis $p = 1$.
- Case 2: Only v has $2t + 1$ next-votes for period $p - 1$. Then in period p, every honest node will have starting value $st_i^p = v$, so only v could receive soft-vote and next-vote.

- Case 3: Both $\perp$ and v have $2t+1$ next-votes for period $p-1$. Suppose a new value $v' \neq v$ receives $2t+1$ next-votes for period p, then v' must have $2t+1$ soft-votes for period p and at least $t+1$ soft-votes are from honest nodes, which means that at least $t+1$ honest nodes have received $2t+1$ next-votes for $\perp$ for period $p-1$ when they enter Step 2. Hence when they enter Step 4, they will next-vote v' or $\perp$ instead of v. And they will not next-vote v in Step 5 either. Then v can have at most t next-votes from honest nodes and at most $2t$ next-votes from all nodes for period p.

Therefore, we proved the claim for period p. Then we prove the lemma. Note that a node will stuck in period p if and only if no value receives at least $2t+1$ next-votes for period p. Then by the previous claim we proved, there are only two possible cases:

- Case 1: $\perp$ has $2t+1$ next-votes for period $p-1$. If no value has at least $2t+1$ soft-votes for period p, then every honest node will just next-vote $\perp$ in Step 4 or Step 5 (since $2t+1$ next-votes on $\perp$ for period $p-1$ will eventually reach every honest node). Otherwise if some value v has at least $2t+1$ soft-votes for period p, then every honest node will next-vote v.

- Case 2: Only some value v has $2t+1$ next-votes for period $p-1$. Then in period p, every honest node will have starting value $st_i^p = v$, so only v could receive soft-vote and next-vote. And every honest node will next-vote v in Step 4.

□

The original Algorand BA guarantees fast recovery from network partition: after GST, an agreement is reached in $O(\Delta)$ time. However, our protocol does not have such a strong liveness property as honest nodes may disagree on what is the longest chain for some time right after GST. Here in Lemma 3.3, we prove that our protocol is live and each iteration takes $O(\Delta)$ time to finish once the common prefix property of all honest nodes' longest chains is recovered, which may take $O(\mathsf{GST})$ time as we will see in Section 3.8.

Lemma 3.3 (Conditional Liveness After GST, **CP3**)**.** *Suppose after time T, all honest nodes always have the same values, i.e., their local checkpointed longest chains satisfy common prefix property, then after* $\max\{\mathsf{GST}, T\}$*, each iteration of Algorand BA will finish within $O(\Delta)$ time. Moreover, all honest nodes finish the same iteration within time Δ apart.*

Proof. Let N be the largest iteration number such that some honest node has committed some value v before GST. Then all honest nodes will receive the certificate for v within time Δ after GST and they will all enter iteration $N+1$ after some time. If such N does not exist, i.e., no value has been committed by any honest node before GST, then we set $N = 0$.

Further, let p be the highest period that some honest node is working on in iteration $N+1$ right before GST is. Then after time Δ, all honest nodes will also start period p as they receive $2t+1$ next-votes for period $p-1$. Soon after, some value v (which may be $\perp$) will receive at least $2t+1$ for period p by Lemma 3.2, and all honest node will all start period $p+1$ within time Δ apart.

Once all honest nodes start the same period p within time Δ apart, the network partition seems to have never happened. Without partition, if all honest nodes have the same value, i.e., after $\max\{\mathsf{GST}, T\}$, the following facts hold:

- By Lemma 3.2, there exist at most two values each of which has $2t+1$ next-votes for every period, and one of them is necessarily $\perp$ if there exist exactly two.

- If a period p is reached and the leader is honest, then all honest nodes finish the current iteration in period p by their own time 6Δ, with the same output $v \neq \perp$. Moreover, all honest nodes finish within time Δ apart. Indeed, if there exists a potentially committed value v for period $p-1$, then v will be the unique potential starting value for period p. In period p, the honest leader proposes v in Step 1 and all honest nodes soft-vote v in Step 2. In Step 3, by their own time 4Δ, all honest nodes have cert-voted v. Thus all of them finish the current iteration by their own time 6Δ, with output v. Otherwise if there is no potentially committed value in period $p-1$, then the leader of period p may propose its private input v' or a value $v \neq \perp$ for which it has seen $2t+1$ next-votes from period $p-1$. In the first case, all honest nodes will soft-vote v' in Step 2 as they all see $2t+1$ next-votes for $\perp$ from period $p-1$; in the second case, all honest nodes will soft-vote v in Step 2 either by condition 1 or condition 2. In both cases, all honest nodes will cert-vote the same value in Step 3 by their own time 4Δ and finish the current iteration with that value by their own time 6Δ.

- If a period p is reached and there is no committed value for period p (which only happens if the leader is malicious), then all honest nodes move to period $p+1$ by their own time 8Δ, and they move within time Δ apart. Note that the honest nodes may start period p not at the same time but within time Δ apart. Indeed, if no honest node has cert-voted in Step 3, then all honest nodes next-vote a common starting value v in Step 4 if no one has seen $2t+1$ next-votes for $\perp$ for period $p-1$, or else all honest nodes next-vote $\perp$ in Step 4 or Step 5. In both cases, they all see these next-votes and move to period $p+1$ by their own time 8Δ. If at least one honest node has cert-voted a value v in Step 3, then v must have received $2t+1$ soft-votes and no other value could have these many soft-votes by quorum intersection. Thus only v could have received cert-votes. Since those honest nodes have helped rebroadcasting the soft-votes for v by their own time 4Δ, all honest nodes see $2t+1$ soft-votes for v by their own time 6Δ. Thus all honest nodes next-vote v either in Step 4 or Step 5 by their own time 6Δ, and all see $2t+1$ next-votes for the same value by their own time 8Δ. Note that some honest nodes may have next-voted for $\perp$ in Step 4 as well, thus there may also exist $2t+1$ next-votes for $\perp$.

- Combining the above facts together, since the leader in each period is honest with probability $> 2/3$, one iteration takes in expectation at most 1.5 periods and at most 10Δ time. Moreover, all honest nodes finish within time Δ apart.

□

The liveness property showed in Lemma 3.3 conditions on the assumption that all honest nodes will have the same values after some time T. However, this assumption may not hold right after GST as the adversary could have a bank of private blocks that can be used to break the common prefix property. During this stage, Algorand BA may still output some values in favor of the adversary (eg., the adversary sends its favorite chain to all honest nodes). But if the output value is too old in the sense that no honest node has that checkpointed block in its local longest chain for a long time, then all honest blocks mined during this time interval could be potentially wasted as they are not extending the last checkpoint. However, we show that Algorand BA with our cautious modification won't let this happen in the following lemma.

Lemma 3.4 (Recency Condition After GST, **CP1**)**.** *After GST, if one honest node commits a value v in one iteration at time t, then v must have been the input value held by at least one honest node at time no earlier than $t - 8D\Delta$, where $D \sim$ Geo(p) with $p > 2/3$ being the fraction of honest nodes.*

Proof. For a fixed iteration, We define V_p to be the set of potential starting values for period p. Then $V_1 = \{\perp\}$ and also by Lemma 3.2, we have $1 \leq |V_p| \leq 2$, and $\perp \in V_p$ if $|V_p| = 2$ for all p. Note that $V_p = \{v, \perp\}$ is an bad set as v may be an outdated value. Then we show that the state won't remain in $\{v, \perp\}$ for the same v for a long time.

Suppose the leader of period p is honest and no honest node has value v for period p, then we have the following two cases.

- Case 1: if the leader of period p has received $2t + 1$ next-votes for $\perp$ for period $p - 1$ at the beginning of Step 1, then it will propose $v' \neq v$. Further no honest node will soft-vote or next-vote v. Therefore, the state of period $p + 1$ will be $\{\perp\}$, $\{v'\}$, or $\{v', \perp\}$.
- Case 2: if the leader of period p has received $2t + 1$ next-votes only for v for period $p - 1$ at the beginning of Step 1, then it will propose v. Further every honest node will soft-vote v in Step 2 either by condition 1 or condition 2. Therefore, v will be committed in period p.

Therefore, after GST, the state won't remain in $\{v, \perp\}$ for the same v from period p to period $p + 1$, if the leader of period p is honest and no honest node has value v for period p. By Lemma 3.3, we know that after GST each period (even with a malicious leader) will finish in 8Δ time. Combining these facts together, we can conclude the lemma.

□

We say the i-th checkpoints committed at some time t by an honest node after GST is *D_i-recent* if it has been held at depth exactly k in the chain of an honest node at some time $t' \geq t - D_i$. Then by Lemma 3.4, we have D_i is stochastically dominated by $8D\Delta$, where $D \sim$ Geo(p) with $p > 2/3$. Let us define event E to be the event that all checkpoints committed after GST and before $T_{\max}$ are d-recent, where d is called the recency parameter of the protocol. Then we have the following corollary directly from Lemma 3.4.

Corollary 3.1. *There exists* $d = O(\sqrt{\kappa}\Delta)$ *such that* $\mathbb{P}(E) \geq 1 - e^{-\Omega(\sqrt{\kappa})}$*, where* κ *is the security parameter.*

Proof. Recall that there can be at most $T_{\max}/e$ checkpoints by **CP2**. By the union bound, we have

$$\mathbb{P}(E^c) \leq \sum_i \mathbb{P}(D_i > d) \leq \frac{T_{\max}}{e}\mathbb{P}(\text{Geo}(p) > \frac{d}{8\Delta}) = \frac{T_{\max}}{e}(1-p)^{\frac{d}{8\Delta}} = e^{-\Omega(\sqrt{\kappa})}.$$

□

Thus we conclude that the modified Algorand BA, given in Section 3.6, provides the desired checkpointing properties **CP0-CP3**.

3.8 Checkpointed Longest Chain Properties

3.8.1 Comparison With Longest Chain Protocol

The security properties of the longest chain protocol have been intensely studied in recent years. The strong security properties have been demonstrated in increasing sophistication (both network models as well as tightness of security threshold): the pioneering work on the round by round network model [4] has been extended to discrete and finely partitioned network model [6] and to a continuous time network model [30]. The tightness of the security threshold has been improved to the best possible in [7]. Despite this wealth of technical background, the *checkpointed* longest chain protocol has seemingly subtle differences with the vanilla longest chain protocol, but impact the analysis significantly. We discuss two of these issues next.

Synchrony versus Partial Synchrony

The security analyses of the longest chain protocol require that synchronous network conditions hold from the beginning of the protocol [4, 6, 7]. In the partially synchronous model, where messages may be arbitrarily delayed until some unknown time GST, the protocol's security breaks down. More precisely, even after (a bounded time beyond) GST, safety and liveness may fail. We describe a possible attack scenario in Figure 3.6. The aforementioned works do not provide any security guarantees in the partially synchronous model. However, the method of analysis suggests that it is possible to provide some security guarantees *after* $O(\mathsf{GST})$ *time*. This should hold for any adversarial power β in which the protocol is secure in the synchronous regime. Consider the feasibility of a private attack, in which the adversary must mine a longer chain than that mined by the honest nodes. The adversary has some initial advantage immediately after GST by means of some private blocks (see Figure 3.6), but it will eventually lose out to the honest chain as the latter's growth rate is strictly larger. The work of Dembo et al. [7] shows that the private attack

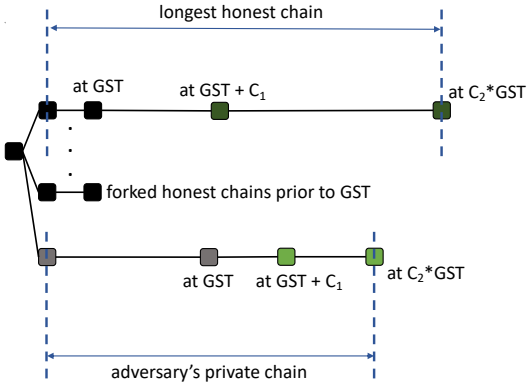

Figure 3.6: An illustration of the private attack by an adversary in a partially synchronous system. The private chain is longer than the longest honest chain for a short while after GST, but not indefinitely after.

is the worst-case attack. To the best of our knowledge, only Neu et al. [27] analyze a longest chain protocol (in particular, the sleepy consensus protocol [23]) in the partially synchronous setting, and they show desirable properties hold after $O(\mathsf{GST})$ time, with the constant being approximately $1/(1-2\beta)$.

Non-Monotonic Chain Growth

The checkpointed longest chain protocol can behave very differently compared to the classical longest chain protocol when checkpoints appear. In the classical version, the chain length of each honest node grows monotonically. In contrast, in our protocol, the chain held by an honest node may reduce in length once it hears of a new checkpoint. This could happen if its current chain does not contain the new checkpoint. This has important ramifications for security. A key property used in analyzing the classical protocol is that (after GST), two honest blocks that appear more than Δ time apart must be at different heights [30]. This property needn't hold for the checkpointed longest chain protocol; in fact, it can be violated if the adversary chooses to do so. In effect, this causes a bleeding (or wastage) of honest mining power. An instance of this is illustrated in Figure 3.7. The analysis by Neu et al. [27] uses many properties of longest chain protocols 'off the shelf' from the work of Pass and Shi [23]. However, for reasons highlighted above, we must establish new properties that hold deterministically for our protocol, irrespective of the adversary's actions. These properties are weaker analogs of those satisfied by the longest chain protocol. This is to be expected, since the checkpointed longest chain protocol may behave just as the longest chain protocol irrespective of GST, if the adversary so wishes.

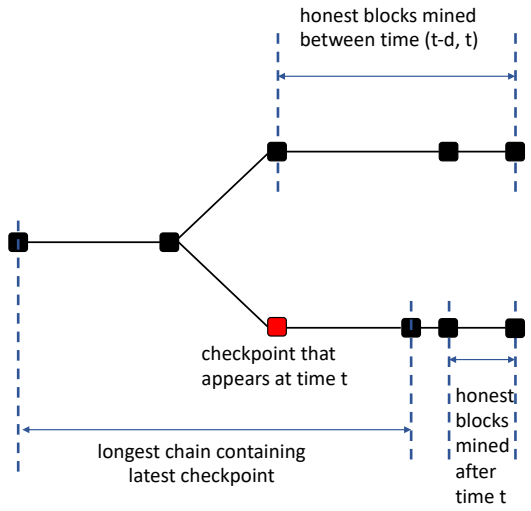

Figure 3.7: An illustration of a reduction in chain length due to the appearance of a checkpoint. Consequently, two honest blocks may be at the same height, even if they are mined more than Δ time apart.

3.8.2 Definitions

We first define the notion of common prefix and chain quality, which are now standard in the literature [4].

Definition 3.6 (k-Common Prefix)**.** *For a chain $\mathcal{C}$, let $\mathcal{C}\lfloor k$ denote the new chai nobtained by dropping the last k blocks from $\mathcal{C}$. We say that the k-common prefix property holds if for any two chains $\mathcal{C}_1$, $\mathcal{C}_2$ held by honest nodes at times $t_1 \leq t_2$ respectively, it holds that $\mathcal{C}_1\lfloor k \preceq \mathcal{C}_2$.*

Definition 3.7 ((k, s)-Chain Quality)**.** *We say that the (k, s)-chain quality property holds if for all honestly held chains $\mathcal{C}$, every portion of $\mathcal{C}$ with k consecutive blocks mined after time s contains at least one honest block.*

Our definition of k-common prefix is identical to the one given by Garay et al. [4]. Our definition of (k, s)-Chain Quality with $s = 0$ is identical to their definition of chain quality with parameter $\mu = 1/k$. We introduce the extra parameter s to emphasize that the chain quality property holds only for the portion of the chain consisting of blocks mined after time $s = O(\mathsf{GST})$. In the partially synchronous model, we cannot hope to do better. Garay et al. [4], show that common prefix is identical to safety (also called persistence) and chain quality is identical to liveness for the k-deep confirmation rule. Thus, to prove security of Π_{ada} after $O(\mathsf{GST})$, it suffices to show that k-common prefix and (k, s)-chain quality hold after time $O(\mathsf{GST})$, for $s = O(\mathsf{GST})$.

3.8.3 Proof Sketch

In this section, we analyze the security of the protocol by discretizing time into slots. Note that this is purely a proof strategy; we do not change any assumptions about the behavior of the nodes, nor of the message timings. Our proof is structured similar to that of Garay et al. [4]. It has three major parts that we describe below.

- First, we define the notion of a typical execution (Definition 3.8). A typical execution is one in which the number of mined honest and adversarial blocks over a sufficiently large period do not deviate from their mean by a large fraction. We introduce the notion of a *convergence opportunity* here [6]. We show that a typical execution occurs with high probability (Corollary 3.2).

- Next, we show necessary conditions for common-prefix violation and chain quality violation in terms of the mining process. These conditions are analogous to the condition in the classical longest chain protocol, which are roughly as follows: the number of adversarial blocks over an interval is more than the number of honest blocks over the same interval.

- Third, we show that these necessary conditions won't happen after $O(\mathsf{GST} + \kappa)$ time in a typical execution (Lemma 3.10). Hence, with high probability, the common prefix property and the chain quality property of the checkpointed longest chain will hold after $O(\mathsf{GST} + \kappa)$ time (Corollary 3.3).

The first part is similar to existing works. Our main novelty lies is the second and third parts. As hinted above, we can at best hope to obtain weaker necessary conditions for common prefix violation and chain quality violation than those for the classical protocol. At the same time, a condition that is too weak may not be rare in a typical execution. We show that the necessary conditions are very similar for both k-common prefix violation and (k, s)-chain quality violation. Moreover, we show that these conditions are impossible in a typical execution after $O(\mathsf{GST} + \kappa)$ time. Our analysis relies heavily on the properties of the checkpointing protocol proved in Section 3.7. For clarity, we further split this part of the proof into four steps.

- We reduce all common-prefix violation instances to a special case where two diverging chains are held by honest nodes at the same slot.

- We establish a lower bound on the lengths of honestly held chains in terms of convergence opportunities in an interval, with some slack to account for the possibility of bleeding.

- We establish the desired necessary condition for k-common prefix violation by analyzing the common portion and the forked portion of the two diverging honestly held chains.

- We establish the desired necessary condition for (k, s)-chain quality violation by comparing the chain growth for honest chains in an interval with the number of adversarial blocks in the same interval.

In this section, whenever we refer to a block B being k-deep in a chain $\mathcal{C}$, we mean there are k or more blocks that are descendants of B in $\mathcal{C}$.

3.8.4 Typical Execution

From Section 3.4, we know that the block mining process is a Poisson process with rate λ. This can be further split into an adversarial block mining process of rate $\beta\lambda$ and an independent honest block mining process of rate $(1-\beta)\lambda$, with $\beta < 1/2$. Therefore the number of honest/adversarial blocks mined in any interval is a Poisson random variable. By the Chernoff bound, we know that a Poisson random variable is tightly concentrated around its mean. However, we would like to obtain such a result for *all intervals* of length bigger than $O(\kappa)$. Hence, for the sake of applying the union bound, we discretize time into slots: the i^{th} slot is the time interval $((i-1)\Delta, i\Delta]$. The total number of slots in the entire execution of the system is $t_{\max} = \lceil T_{\max}/\Delta \rceil = O(\text{poly}(\kappa))$. We abuse notation to let GST denote the first slot that begins after the global synchronization time. All our analysis henceforth will be in the form of slots, but the protocol remains a continuous time one as described in Section 3.4.

We define the following random variables for slot i.

- $Y_i = 1$ if there is exactly one block that is mined by honest nodes in slot i and there is no block that is mined by honest nodes in slot $i-1$ and $i+1$, otherwise 0. We follow the notation in [6] to call such slots as a *convergence opportunity* (such a block is also called *loner* in [30]).
- Z_i is the number of adversarial blocks plus the number of honest blocks that are not a convergence opportunity, in slot i.

Let

$$\bar{y} \triangleq \mathbb{E}[Y_i] = (1-\beta)\lambda\Delta\ e^{-3(1-\beta)\lambda\Delta}.$$

Further, $\mathbb{E}[Y_i + Z_i] = \lambda\Delta$, and hence

$$\bar{z} \triangleq \mathbb{E}[Z_i] = \lambda\Delta - \bar{y}.$$

For any $t_1 \leq t_2$, we define $Y[t_1, t_2] = \sum_{i=t_1+1}^{t_2} Y_i$ and $Z[t_1, t_2] = \sum_{i=t_1+1}^{t_2} Z_i$. With some abuse of notation, we let $Y[S]$ denote $\sum_{i\in S} Y_i$ for any set of slots $S \subset \mathbb{N}$. As we are now in discrete time, the notation $[t_1, t_2]$ denotes the set of slots $\{t_1, \ldots, t_2\}$. We have that

$$\mathbb{E}Y[t_1, t_2] = \bar{y}(t_2 - t_1), \quad \mathbb{E}Z[t_1, t_2] = \bar{z}(t_2 - t_1).$$

We note that for any $\beta < 1/2$ and any Δ, there exists $\varepsilon > 0$ and $\lambda > 0$ s.t.

$$\bar{y}(1 - 3\varepsilon) > \bar{z}.$$

In particular, this implies $\bar{y} > \bar{z}$, and $(1-\varepsilon)\bar{y} > \bar{z} + \varepsilon\bar{y}$. We shall operate under this regime. We also assume that $\lambda\Delta < 1$. This condition implies $\bar{y} = \Omega(\lambda\Delta)$, where the constants hidden by $\Omega(\lambda\Delta)$ do not depend on β, λ, Δ.

With these notations in place, we can now define the notion of a *typical execution*.

Definition 3.8 (Typical execution)**.** *An execution is (ε, τ)-typical for a β-corrupt system, for $\varepsilon \in (0,1)$ and $\tau \in \mathbb{N}$, if the following events hold for any t_1, t_2 s.t. $t_2 - t_1 \geq \tau$:*

$$|Y[t_1,t_2] - \mathbb{E}Y[t_1,t_2]| \leq \varepsilon \mathbb{E}Y[t_1,t_2];$$
$$|Z[t_1,t_2] - \mathbb{E}Z[t_1,t_2]| \leq \varepsilon \mathbb{E}Y[t_1,t_2];$$
$$|Y[t_1,t_2] + Z[t_1,t_2] - (\mathbb{E}Y[t_1,t_2] + \mathbb{E}Z[t_1,t_2])| \leq \varepsilon(\mathbb{E}Y[t_1,t_2] + \mathbb{E}Z[t_1,t_2]).$$

Proposition 3.1. *For any $t_1 \leq t_2$, the following events hold with probability at least $1 - e^{-\Omega(\varepsilon^2\lambda\Delta|t_2-t_1|)}$:*

$$|Y[t_1,t_2] - \mathbb{E}Y[t_1,t_2]| \leq \varepsilon \mathbb{E}Y[t_1,t_2];$$
$$|Z[t_1,t_2] - \mathbb{E}Z[t_1,t_2]| \leq \varepsilon \mathbb{E}Y[t_1,t_2];$$
$$|(Y[t_1,t_2] + Z[t_1,t_2]) - (\mathbb{E}Y[t_1,t_2] + \mathbb{E}Z[t_1,t_2])| \leq \varepsilon(\mathbb{E}Y[t_1,t_2] + \mathbb{E}Z[t_1,t_2]).$$

Proof. The Chernoff bound for a binomial or a Poisson random variable X gives us

$$\mathbb{P}(|X - \mathbb{E}X| \geq \delta \mathbb{E}X) \leq 2e^{-\delta^2 \mathbb{E}X/3}$$

We first prove the concentration bound on $Y[t_1, t_2]$. We cannot directly apply standard concentration inequalities on $Y[t_1, t_2]$ since Y_i's are not independent random variables. However, for any i, Y_i and Y_{i+3} are independent. Each Y_i is Bernoulli($\bar{y}$). Hence, $Y[t_1, t_2]$ can be written as the sum of three binomial random variables.

$$Y[t_1,t_2] = \sum_i Y_{t_1+3i} + \sum_i Y_{t_1+3i+1} + \sum_i Y_{t_1+3i+2}.$$

Using the Chernoff bound and the union bound, we get that each of these three binomial random variables are within a factor of $\varepsilon/3$ from their mean, except with probability with probability $e^{-\Omega(\varepsilon^2\lambda\Delta(t_2-t_1))}$. When this happens, the desired bound on $Y[t_1, t_2]$ follows.

Next, we note that $Y[t_1, t_2] + Z[t_1, t_2]$ is a Poisson random variable. Using the Chernoff bound for Poisson random variables, and the fact that $\mathbb{E}(Y[t_1, t_2] + Z[t_1, t_2]) = \lambda\Delta(t_2 - t_1)$, we obtain that $Y[t_1, t_2] + Z[t_1, t_2]$ is within a factor of ε from its mean except with probability $e^{-\Omega(\varepsilon^2\lambda\Delta(t_2-t_1))}$.

It remains to bound $Z[t_1, t_2]$. For this, we need a slightly tighter bound on $Y[t_1, t_2]$ and $Y[t_1, t_2] + Z[t_1, t_2]$ than the one in the statement of the proposition; we replace ε by $\varepsilon/3$. Both these bounds also hold except with probability $e^{-\Omega(\varepsilon^2\lambda\Delta(t_2-t_1))}$. From these bounds, and the relation $\bar{y} > \bar{z}$, we can the desired bound on $Z[t_1, t_2]$. □

There are at most $t_{\max}^2$ choices of t_1, t_2. The following corollary follows from Proposition 3.1 and

the union bound.

Corollary 3.2. *The protocol executes with (ε, τ)-typical execution with probability at least $1 - e^{-\Omega(\varepsilon^2\kappa)}$.*

Proof. From Proposition 3.1 and the union bound, we get that the probability of a typical execution is at least $1 - t_{\max}^2 e^{-\Omega(\varepsilon^2\lambda\Delta\tau)}$. We can choose τ large enough such that $\lambda\Delta\tau = \Theta(\kappa)$. We know that $t_{\max} = \text{poly}(\kappa)$. From this, the stated result follows. □

3.8.5 Necessary Conditions for Common Prefix/Chain Quality Violation

We begin by noting some basic properties of the checkpointing protocol and also introduce some new notation that is useful for this subsection. Our first result is a chain growth lemma (Lemma 3.6). The chain growth lemma highlights the extent of bleeding that can take place in the checkpointing protocol. Next, we focus on the common prefix property. We show that without loss of generality, we can assume that common-prefix violation occurs between two chains at the same slot (Lemma 3.7). Following this, we show that in the forked portion, the number of adversarial blocks over the forked portion must be at least half the total number of blocks. This gives us the desired necessary condition comparing the number of convergence opportunities and adversarial blocks over a sufficiently large period (Lemma 3.8). Finally, we analyze the chain quality property in Lemma 3.9.

Basic properties of checkpoints

Let us recall some properties of the checkpointing algorithm, re-stated in terms of slots instead of discrete time.

1. We say that a checkpoint appears in slot t if the first time an honest node marks the checkpoint is in slot t. If a checkpoint appears at slot t, all honest nodes mark the checkpoint by the beginning of slot $t + 2$.
2. Let $d' \triangleq \lceil d/\Delta \rceil$. Any checkpoint that appears at slot $t > \mathsf{GST} + d'$ was at least k-deep in a chain held by an honest node in some slot $t' \in \{t - d', \ldots t\}$.
3. Let $e' = \lfloor e/\Delta \rfloor$. If two consecutive checkpoints appear at slots t, t', then $t' \geq t + e'$.

Let $\mathsf{GST} + d' < t_1^* \leq t_2^* \leq \ldots$ be the sequence of slots at which checkpoints appear after slot $\mathsf{GST} + d'$. Let the corresponding checkpoint blocks be $B_1^*, B_2^*, \ldots$. Let $t_0^* \triangleq \mathsf{GST}$. Define $S_l \triangleq \{t_l^* + 2, \ldots, t_{l+1}^* - d' - 2\}$ for all $l \geq 0$ and let $S \triangleq \cup_{l \geq 0} S_l$. We refer to each S_l as an inter-checkpoint interval.

Chain growth results

Lemma 3.5 (Chain Length Lower Bound)**.** *Let t_1, t_2 be two slots such that $t_1 \in S$ and and $t_2 \geq t_1 + 2$. Let $\mathcal{C}$ be a chain held by some honest node in slot t_1. Then all honest nodes will hold a chain of length at least $|\mathcal{C}|$ in slot t_2.*

Proof. Let h be an honest node that holds chain $\mathcal{C}$ in slot t_1, and let h' be an arbitrary honest node (possibly h) that holds a chain $\mathcal{C}'$ in slot t_2. Since $t_1 \in S$, we know that $t \in S_l$ for some l. Therefore B_l^* is the last checkpoint that is marked in $\mathcal{C}$. Clearly, all honest nodes will hear of $\mathcal{C}$ by the beginning of slot t_2, and would also have marked B_l^* as a checkpoint. We aim to show that $|\mathcal{C}'| \geq |\mathcal{C}|$. There are now two cases to consider.

- Node h' has not marked any new checkpoint at the time it holds $\mathcal{C}'$. In this case, it will hold a chain at least as long as $\mathcal{C}$, as it is a chain that it has heard of that contains all the checkpoints that it has marked.

- Node h' has marked at least one new checkpoint at the time it holds $\mathcal{C}'$. In particular, it has marked B_{l+1}^*. Note that B_{l+1}^* must be exactly k-deep in some honestly held chain $\mathcal{C}^*$ at some slot in the interval $[t_{l+1}^* - d', t_{l+1}^*]$ in which B_{l+1}^* was not marked as a checkpoint. By the first case, $\mathcal{C}^*$ must be at least as long as $\mathcal{C}$, since $t_1 \leq t_{l+1}^* - d' - 2$. Further, all honest nodes must hold a chain at least as long as $\mathcal{C}^*$ at the time of marking B_{l+1}^*. Arguing inductively, we see that all honest nodes that have marked at least one new checkpoint compared to $\mathcal{C}$ also hold a chain at least as long as $\mathcal{C}$.

□

Lemma 3.6 (Chain Growth)**.** *Consider a chain $\mathcal{C}$ held by an honest node in a slot $t \geq \mathsf{GST} + 1$. Suppose $\mathcal{C}$ contains an honest block B mined in slot s at height ℓ. Then $|\mathcal{C}| \geq \ell + Y[S \cap [s+2, t-2]]$.*

Proof. From Lemma 3.5, we can deduce three useful facts. Firstly, the first converge opportunity block in $S \cap [s+2, t-2]$ will be built at a height $\geq \ell + 1$, since all honest nodes by that slot would have adopted a chain at least as long as ℓ. Secondly, we see that all convergence opportunities in S are mined at distinct heights. This is because all convergence opportunities are spaced at least 2 slots apart. Thirdly, all honest nodes will adopt a chain that is at least as long as the chain mined by the last converge opportunity in $S \cap [s+2, t-2]$. Together, these two statements prove the desired statement when $s \in S$.

To prove the same result when $s \notin S$, we consider the following two cases:

- Suppose $s \notin S$, and $s \geq \mathsf{GST}$. This implies $s \in \{t_l^* - d' - 1, \ldots t_l^* + 1\}$ for some $l \geq 1$. then in slot $t_l^* + 2$ all honest nodes will see both block B and checkpoint B_l. Moreover, block B and checkpoint B_l are on the same chain $\mathcal{C}^*$, hence all honest nodes should hold a chain of length at least ℓ in slot $t_l^* + 2$, simply because $\mathcal{C}^*$ contains B. The rest of the proof is identical to the case $s \in S$.

- Suppose $s \notin S$ and $s < \mathsf{GST}$. Let $\mathcal{B}$ be the set of checkpoints with appearance time <= GST, i.e, for a block $B^* \in \mathcal{B}$, at least one honest node marks B^* before GST. Then in slot $\mathsf{GST}+1$, all honest nodes will see a chain $\mathcal{C}^*$ that contains block B and all checkpoints in $\mathcal{B}$. Hence all honest nodes should hold a chain of length at least ℓ in slot $\mathsf{GST}+1$, simply because $\mathcal{C}^*$ contains B. The rest of the proof is identical to the case $s \in S$.

□

Analysis of Common Prefix Property

Let us first note a simple fact about the checkpointed longest chain rule. In what follows, we say a chain $\mathcal{C}$ was held by an honest node in slot t if it was held by the node at any time in the interval $((t-1)\Delta, t\Delta]$.

Proposition 3.2. *If a block B is k-deep in a chain $\mathcal{C}$ held by an honest node h in slot t, then it either remains k-deep in $\mathcal{C}'$ or is not present in $\mathcal{C}'$, where $\mathcal{C}'$ is an chain held by h in slot t'.*

The above statement is trivially true for the longest chain rule, since the chain length held by an honest node is monotonically increasing. It is not so immediate for the checkpointed longest chain rule, since the length of the chain held by a particular honest node may reduce upon adopting a new checkpoint block. A crucial point in the proof of this statement is the property that a checkpoint is always k-deep in all honest nodes' chains. The statement is illustrated in Figure 3.8, and is proven below.

Proof. As remarked above, the property holds trivially as long as the honest node h adopts successive chains of increasing length. Consider the case when at some slot $t' \geq t$, h adopts a new checkpoint block and truncates its chain in the process. Suppose block B is present in both $\mathcal{C}$ and $\mathcal{C}'$. Then it must be an ancestor of the new checkpoint block. Since the new checkpoint block is at least k-deep in the new chain, so is block B. □

Lemma 3.7 (Common Prefix Reduction)**.** *Suppose the common prefix property is violated after slot t_0, i.e., there exist slots t_1 and t_2: $t_0 \leq t_1 \leq t_2$, honest nodes h_1, h_2 (possibly the same), and chains $\mathcal{C}_1$, $\mathcal{C}_2$, s.t. $\mathcal{C}_1$ is held by h_1 at slot t_1, $\mathcal{C}_2$ is held by h_2 at slot t_2, but $\mathcal{C}_1\lfloor k \not\preceq \mathcal{C}_2$. Then the following event happens at some slot $t \in [t_1, t_2]$: there exists chain $\mathcal{C}$ and $\mathcal{C}'$ such that $\mathcal{C}'\lfloor k \not\preceq \mathcal{C}$ while both $\mathcal{C}'$ and $\mathcal{C}$ are held by honest nodes in slot t.*

We refer to this events as *common prefix violation at slot t*, and the pair of chains $\mathcal{C}'$ and $\mathcal{C}$ as a *certificate of the violation*. Lemma 3.7, stated for the checkpointed longest chain rule, is also true for the longest chain rule. The proofs are identical as well, given that the statement of Lemma 3.2 holds for both protocols. We provide the proof for completeness.

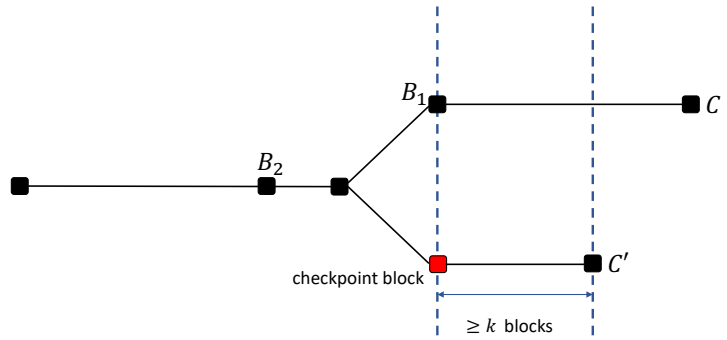

Figure 3.8: Illustration of Lemma 3.2. Both B_1, B_2 are k-deep in $\mathcal{C}$. B_1 gets ousted in $\mathcal{C}'$, while B_2 remains k-deep.

Proof. Let $t \geq t_1$ be the earliest slot at which there exists some chain $\mathcal{C}$ held by an honest node h (possibly h_1) s.t. $\mathcal{C}_1\lfloor k \not\preceq \mathcal{C}$. Let B be the block that is exactly k-deep in $\mathcal{C}_1$; clearly, it is not present in $\mathcal{C}$. We now claim that there must be a chain $\mathcal{C}'$ held by node h_1 at some time in slot t that contains B. Note that $\mathcal{C}$ and $\mathcal{C}'$ satisfy the relation $\mathcal{C}'\lfloor k \not\preceq \mathcal{C}$, thus proving the desired statement. It remains to show the claim.

Suppose h_1 never holds a chain containing B in slot t. Then it must have adopted a chain that does not contain B in an earlier slot. However, we have defined t to be the earliest slot such that some chain held by an honest node does not contains B. Thus, we have shown the claim to be true. A technical point to note is that if an honest node adopts a new chain $\mathcal{C}_{\text{new}}$ at time s in slot t, it must have held the older chain $\mathcal{C}_{\text{old}}$ at some time $(s - \epsilon)$ *in slot* t *itself.* □

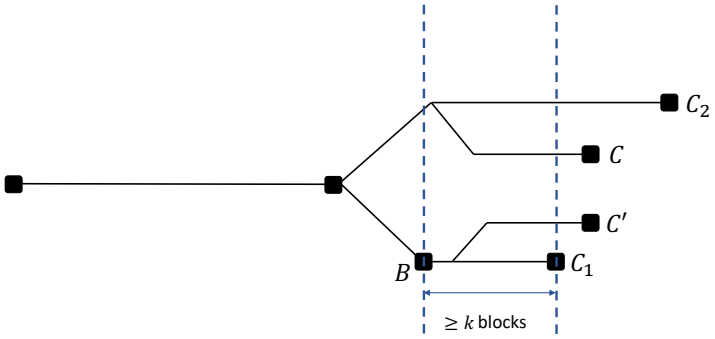

Figure 3.9: Illustration of Lemma 3.7. Block B is k-deep in $\mathcal{C}_1$. It remains so in $\mathcal{C}'$. However, it is absent from $\mathcal{C}$. In this case, $\mathcal{C}'$ and $\mathcal{C}$ form a certificate of k-common prefix violation.

Lemma 3.8 (Necessary condition for common prefix violation)**.** *Suppose k-common prefix violation occurs at slot t. Then there exist $t^* < t$ such that $Z[t^*, t] + Y[t^*, t] \geq k$ and $Z[t^*, t] \geq Y[S \cap [t^* + 2, t - 2]] - 1$.*

Proof. The first condition $Z[t^*, t] + Y[t^*, t] \geq k$ is trivial, as a certificate of k-common prefix violation contains at least k blocks. Then we focus on the derivation of the second condition. From Lemma 3.7, we know that there exists two honestly held chains $\mathcal{C}, \mathcal{C}'$ such that $\mathcal{C}' \lfloor k \not\preceq \mathcal{C}$. Let ℓ be the height of the shorter of the two chains. Consider the last block on the common prefix of $\mathcal{C}, \mathcal{C}'$ that is a convergence opportunity. Denote it by $\overline{B}$, let its height be $\overline{\ell}$ and let $\overline{t}$ be the slot at which it was created. If there is no convergence opportunity in the common prefix, let $\overline{B}$ be the genesis block and $\overline{t} = 0$. From Lemma 3.6, we know that $\ell \geq \ell^* + Y[S \cap [t^* + 2, t - 2]]$, which implies $\ell - \ell^* \geq Y[S \cap [t^* + 2, t - 2]]$. We now show that $Z[t^*, t] \geq \ell - \ell^* - 1$.

Firstly, we note that any adversarial blocks in $\mathcal{C}, \mathcal{C}'$ that are descendants of $\overline{B}$ must be mined in the interval $[t^*, t]$. Let $\tilde{B}$ denote the last block on the common prefix of $\mathcal{C}, \mathcal{C}'$, let its height be $\tilde{\ell}$. The portion of the common prefix from block $\overline{B}$ (excluded) to $\tilde{B}$ (included) is composed entirely of adversarial blocks. This accounts for $\tilde{\ell} - \overline{\ell}$ adversarial blocks.

We now consider two possibilities regarding the number of checkpoints on the chains $\mathcal{C}, \mathcal{C}'$. In the first case, both chains have the same number of checkpoints marked. In this case, all checkpoints must be on the common prefix of $\mathcal{C}, \mathcal{C}'$. In the second case, one of the chains has one extra checkpoint. This checkpoint must have appeared in slot t or slot $t - 1$, for otherwise, it would be adopted by the other chain too. In both cases, let $\hat{B}$ denote the last *common* checkpoint of the two chains. We now analyze the forked portion of the chains in both cases.

For the first case, we show that all convergence opportunity blocks that are descendants of $\hat{B}$ must be at distinct heights. Note that every chain ending at each such convergence opportunity blocks contain all checkpoints that appear until slot t. Thus, at the time when any honest node would have received such a chain (say $\tilde{\mathcal{C}}$), it would adopt $\tilde{\mathcal{C}}$ or a chain of at least $|\tilde{\mathcal{C}}|$. This is because $\tilde{\mathcal{C}}$ would contain all the checkpoints that this node would have marked until then. Since the miner of every convergence opportunity hears of broadcasts from all previous convergence opportunities before mining, the claimed result follows. For the second case, a slightly modified statement holds: all convergence opportunities that are descendants of $\hat{B}$ mined in slots $\leq t - 2$ must be at distinct heights. This can be proven by arguing in the same manner as above. Since the latter is a weaker claim than the former, we use the latter claim in the general case.

Note that all blocks in the forked portion are descendants of $\tilde{B}$, and are therefore descendants of $\hat{B}$. Therefore, all convergence opportunity blocks in the forked portion mined in slots $\leq t - 2$ must be at distinct heights. Since they can be on at most one of the chains, they must be paired with either an adversarial block or a convergence opportunity from slot $t - 1$ or t. There can be at most one convergence opportunity from slots $\{t - 1, t\}$. From this, we conclude that the number of adversarial blocks in the forked portion is at least $\ell - \tilde{\ell} - 1$.

In summary, we can say that the number of distinct adversarial blocks in chains $\mathcal{C}, \mathcal{C}'$ combined that are descendants of block B^* is at least $\ell - \tilde{\ell} - 1 + \tilde{\ell} - \overline{\ell} = \ell - \overline{\ell} - 1$. All these blocks are mined in slots in the range $[t^*, t]$. Thus, $Z[t^*, t] \geq \ell - \ell^* - 1$. Combined with the inequality $\ell - \ell^* \geq Y[S \cap [t^* + 2, t - 2]]$, we get the inequality: $Z[t^*, t] \geq Y[S \cap [t^* + 2, t - 2]] - 1$. □

Analysis of Chain Quality Property

With some abuse of notation, we use the term (k, s)-chain quality to refer to a slot s instead of some time s.

Proposition 3.3 (Chain Quality Reduction)**.** *Suppose at some slot t, (k, s)-chain quality is violated, i.e., there exists an honestly held chain $\mathcal{C}$ in slot t that contains k-consecutive blocks mined after slot s and there is no honest block amongst them. Then at some slot $t' \in (s, t]$, there exists a chain $\mathcal{C}'$ held by an honest node such that the last k blocks of $\mathcal{C}'$ are all adversarial blocks.*

Proof. Suppose, at some slot t, (k, s)-chain quality is violated. Clearly, $t > s$, else it is not possible for $\mathcal{C}$ to contain blocks mined at slots $> s$. There are two cases to consider. First, we consider the case where there are no more honest blocks in $\mathcal{C}$ after the string of k consecutive adversarial blocks. Then the claim trivially follows with $t' = t$, $\mathcal{C}' = \mathcal{C}$. Next, consider the case where there exists an honest block in $\mathcal{C}$ after the string of k consecutive adversarial blocks. Let the block be B and let its mining time be t'. Since these k blocks are all mined after slot t_0, so is B (i.e., $t' > s$). Then the portion of $\mathcal{C}$ up to B (but not including it) was held by the honest miner of block B. Call this sub-chain $\mathcal{C}'$. From the condition on $\mathcal{C}$, it is immediate that the last k blocks of $\mathcal{C}'$ are adversarial. Thus, we have our claim. □

If there exists an honestly held chain $\mathcal{C}$ at slot t such that the last k blocks of $\mathcal{C}$ are all adversarial blocks mined after slot s, we say that (k, s)-*chain quality violation occurs at slot* t. From Proposition 3.3, we assume without loss of generality that all violations of (k, s)-chain quality are such that (k, s)-chain quality violation occurs at slot t, for some $t > s$.

Lemma 3.9 (Necessary Condition for Chain Quality Violation)**.** *Suppose, for any slot $t \geq \mathsf{GST}+1$, (k, s)-chain quality is violated at slot t. Then there $\exists\ t^* < t$ s.t. $Z[t^*, t] \geq k$ and $Z[t^*, t] \geq Y[S \cap [t^* + 2, t - 2]]$.*

Proof. Let $\mathcal{C}$ be the honestly held chain at slot t for which chain quality is violated. Let t^* be the time of mining of the last honest block in $\mathcal{C}$. Let this block be called B and let its height be ℓ. By Lemma 3.6, we have $|\mathcal{C}| \geq \ell + Y[S \cap [t^* + 2, t - 2]]$. We know that the portion of $\mathcal{C}$ from height ℓ onward is composed only of adversarial blocks. These must be mined at or after slot t^*. Therefore, $|\mathcal{C}| - \ell \leq Z[t^*, t]$. This implies $Z[t^*, t] \geq Y[S \cap [t^* + 2, t - 2]]$. Furthermore, we know that $Z[t^*, t] \geq k$ by the condition that the last k blocks do not contain any convergence opportunity. □

3.8.6 Probabilistic guarantee

We begin with a proposition that transforms the necessary conditions from Lemmas 3.8 and 3.9 to a more convenient form for analysis.

Proposition 3.4. *Suppose k-common prefix violation or (k, s)-chain quality violation occurs at slot t. Then there exist $t^* < t$ such that $Z[t^*, t] + Y[t^*, t] \geq k$ and $Z[t^*, t] \geq Y[\max(\mathsf{GST}, t^*) + 2, t - 2] - (1 + (t - t^*)/e')(d' + 3) - 1$.*

Proof. By comparing the necessary conditions in Lemma 3.8 and Lemma 3.9, it suffices to show that $Z[t^*, t] \geq Y[S \cap [t^*+2, t-2]]-1$ implies $Z[t^*, t] \geq Y[\max(\mathsf{GST}, t^*)+2, t-2]-(1+(t-t^*)/e')(d'+3)-1$. Indeed, $Z[t^*, t] \geq Y[S \cap [t^* + 2, t - 2]] - 1$ implies

$$Z[t^*, t] \geq Y[\max(\mathsf{GST}, t^*) + 2, t - 2] - n(d' + 3) - 1,$$

where n is the number of checkpoints that appear in the interval $[t^*, t]$. We now show a upper bound on n: $n \leq (t - t^*)/e' + 1$. Indeed, a block mined in slot t^* cannot be checkpointed at a slot earlier than t^*. Of course, at slot t, we only see checkpoints that appear until slot t. No more than $\lceil (t - t^*)/e' \rceil \leq (t - t^*)/e' + 1$ checkpoints can appear over such an interval, as consecutive checkpoints are separated at least e' slots apart. Thus, $n \leq (t - t^*)/e' + 1$. Therefore, $Z[t^*, t] \geq Y[\max(\mathsf{GST}, t^*) + 2, t - 2] - (1 + (t - t^*)/e')(d' + 3) - 1$. □

We conclude the proof in this section by showing that the in a typical execution, the condition of Proposition 3.4 does not occur after slot $t > C \cdot \mathsf{GST}$ for some appropriate $C > 0$. Therefore k-common prefix property and (k, s)-chain quality property hold after slot $t > C \cdot \mathsf{GST}$.

Lemma 3.10. *For τ large enough, under (ε, τ)-typical execution, there exist a constant $C > 0$ such that k-common prefix property and (k, s)-chain quality property holds after slot $t > s$, where $k = 2\lambda\Delta\tau$ and $s = C \cdot \mathsf{GST}$.*

Proof. Suppose k-common prefix property or (k, s)-chain quality property is violated at slot $t > C \cdot \mathsf{GST}$, then by Lemma 3.8 we have that there exists $r < t$ such that

$$Z[r, t] + Y[r, t] \geq k, \tag{3.1}$$

and

$$Z[r, t] \geq Y[\bar{r} + 2, t - 2] - (1 + (t - r)/e')(d' + 3) - 1, \tag{3.2}$$

where $\bar{r} \triangleq \max(r, \mathsf{GST})$. We only need to show that this event won't happen under typical execution. If $t - r < \frac{k}{2\lambda\Delta} = \tau$, then under typical execution we have

$$Y[r, t] + Z[r, t] \leq Y[r, r + \tau] + Z[r, r + \tau] \leq (1 + \varepsilon)\lambda\Delta\tau < 2\lambda\Delta\tau = k,$$

which contradicts Eqn. (3.1). Hence we have $t - r \geq \frac{k}{2\lambda\Delta} = \tau$.

Further we can rewrite Eqn. (3.2) as

$$Z[r, t] \geq Y[\bar{r} + 2, t - 2] - (t - r)d_1/e' - d_2, \tag{3.3}$$

where $d_1 = d' + 3$ and $d_2 = d' + 4$. Eqn. (3.3) describes a random walk with drift. Note that the extra terms on the right side of Eqn. (3.3) are small by the fact that $d_2 = O(\sqrt{\kappa}) = O(\sqrt{\tau})$ and e' can be chosen arbitrarily large.

As we have seen $t - r \geq \tau$, then under typical execution we have

$$Z[r,t] \leq (t-r)\bar{z} + \varepsilon(t-r)\bar{y} < (1-3\varepsilon)(t-r)\bar{y} + \varepsilon(t-r)\bar{y} = (1-2\varepsilon)(t-r)\bar{y}.$$

Case 1: $r > \mathsf{GST}$, i.e., $\bar{r} = r$. Then we have

$$Y[r+2, t-2] \geq (1-\varepsilon)(t-r-4)\bar{y}.$$

Therefore, we can always choose e' to be large enough such that $(1-2\varepsilon+d_1/e')(t-r)\bar{y}+d_2 < (1-\varepsilon)(t-r-4)\bar{y}$, given that τ, as well as $t-r$, is large. Therefore, $Z[r,t] < Y[\bar{r}+2, t-2]-(t-r)d_1/e'-d_2$, which contradicts Eqn. (3.3).
Case 2: $r \leq \mathsf{GST}$, i.e., $\bar{r} = \mathsf{GST}$. Then we have

$$Y[\bar{r}+2, t-2] \geq (1-\varepsilon)(t-\mathrm{GST}-4)\bar{y} > (1-\varepsilon)(\frac{C-1}{C}t-4)\bar{y}.$$

Again we can always let e' and C to be large enough such that $(1-2\varepsilon+d_1/e')s\bar{y}+d_2 < (1-\varepsilon)(\frac{C-1}{C}t-4)\bar{y}$, given that τ, as well as t, is large. Then we have $Y[\bar{r}+2, t-2] > Z[0,t]+d_1t/e'+d_2 \geq Z[r,t]+d_1(t-r)/e'+d_2$, which contradicts Eqn. (3.3). □

Then we have the following corollary, which states that the common prefix property and the chain quality property of the checkpointed longest chain will hold after $O(\mathsf{GST})$ time with high probability.

Corollary 3.3. *If we choose k to be large enough such that $k = \Theta(\kappa)$, then there exist a constant $C > 0$ such that k-common prefix property and $(k, C \cdot \mathsf{GST})$-chain quality property hold after slot $C \cdot \mathsf{GST}$ with probability at least $1 - e^{-\Omega(\sqrt{\kappa})}$.*

Proof. All the analysis in this section conditions on that event E occurs (**CP1** holds), and we have $\mathbb{P}(E) \geq 1 - e^{-\Omega(\sqrt{\kappa})}$ by Corollary 3.1. Then combine it with Corollary 3.2 and Lemma 3.10, and apply the union bound, we can conclude the proof. □

3.9 Discussion

In this section, we elaborate on some of the finer details pertaining to our work.

Choice of Algorand as the Checkpointing BFT Protocol We chose Algorand BA as our checkpointing protocol because it satisfies the desired properties **CP0**-**CP3** as listed in Section 3.4. However, some other well-known BFT protocols fail to meet all these properties, and, therefore, we don't use them as a candidate checkpointing protocol. Generally, BFT protocols can be categorized into two classes: 1) Classic BFT protocols, such as PBFT [12] and Hotstuff [33]; 2) Chained BFT protocols, such as Streamlet [47] and Chained Hotstuff [33].

PBFT and Hotstuff don't satisfy the **CP1** (the recency condition). The adversary could potentially lock on a block B privately when it becomes the leader of one view, and then the adversary finalizes B as a checkpoint after a long time when B is no longer in the best chain (B is a descendant of the last checkpoint but not in the longest chain). In this attack, all honest blocks mined during this time period could be potentially wasted as they are not extending B. Further, this resource-bleeding attack could be turned into a perpetual liveness attack in the partial synchronous model. One simple way to fix this bug is by introducing randomness into the protocol: when a new honest leader does not hear any locked block from previous views, it will propose an empty block $\perp$ to refresh the privately locked block, with a certain probability. However, this simple fix introduces unnecessary latency in the normal path when there is no attack and the network is synchronous, as it may need more views with honest leaders to ensure progress.

On the other hand, for Streamlet and Chained Hotstuff, the finalization of one block is supported by its descendants, therefore it is hard to design a checkpointed longest chain protocol such that all finalized checkpoints lie on a single blockchain. Therefore, finding or designing a good checkpointing protocol is non-trivial, and it would be interesting to see whether there is any other BFT protocol that fits our need and perhaps has better latency and lower communication cost.

Checkpointing protocol properties The finality gadget in GRANDPA [37] and the finality layer of Afgjort [36] are also checkpointing protocols. The desired checkpointing properties in GRANDPA are very similar to ours: it has **CP0**, **CP1** and **CP3**, but **CP2** (gap in checkpoints) is missing. Without **CP2**, the mining bleeding could happen frequently, which leads to security vulnerabilities. (Note that GRANDPA also does not have property **P3**, as discussed in Section 3.2).

Afgjort only requires **CP0**, but it has two other desired properties: The "Δ-updated" property guarantees that the chains held by honest nodes are at most Δ blocks beyond the last finalized block; "k-support property" ensures that any finalized block must have been on the chain adopted by at least k honest nodes. The "Δ-updated" property is in the opposite direction of our **CP2**; our protocol satisfies $n/3$-support. We defer a full comparison of all these properties and identifying the minimal set of desired properties to future work.

Attack on GRANDPA In Section 3.2, we claimed that in the GRANDPA design, the k-deep rule is insecure in the synchronous but unsized setting. Here, we describe a possible "roll-back" attack, using some notations from GRANDPA chapter [37]. Let h be the number of online honest voters and we assume $f + 1 \leq h \leq 2f$. Suppose at the beginning of round 1, the genesis block B_0 has two children B_1 and B_2 (they are siblings), where B_1 is an honest block and B_2 is a private block. Then all h honest voters prevote for B_1 in step 3 of round 1. However, all honest voters will get stuck in step 4 if the adversary doesn't send any prevotes, because $g(V_{1,v}) = nil$ and $E_{0,v} = B_0$ hence $g(V_{1,v}) \geq E_{0,v}$ is not satisfied. However, all honest miners will mine on B_2 (assuming tie breaking in favor of the adversary). When B_2 becomes k-deep, the adversary casts enough prevotes

and precommits on B_1 to make B_1 finalized in round 1. Now B_2 is deconfirmed, and all honest miners switch to mine on B_1, creating a *safety violation*. Further if the adversary has mined some private blocks on B_1, then he can continuous this attack in all rounds and all the finalized block will be adversarial block, leading to a *liveness violation*.

3.10 Conclusion

In this chapter, we presented the design of a new finality gadget to be used with Proof-of-Work blockchains. The proposed gadget provides each user a choice between an adaptivity guaranteeing confirmation rule and a finality guaranteeing one. This chapter underscores the fact that it is possible to circumvent the impossibility result suggested by the CAP theorem by appropriately combining the longest-chain protocol with a committee-based BFT protocol. However, this chapter is only a first step towards designing a viable protocol. Several interesting directions of research remain open, which we highlight below.

In our protocol, we used Algorand BA as our checkpointing protocol since it satisfies properties **CP0-CP3**. Other natural candidates, such as PBFT [12], Hotstuff [33] and Streamlet [47] do not satisfy these properties. In particular, they do not satisfy the recency condition **CP1** (we elaborate on this in Section 3.9). It would be interesting to see whether these conditions are necessary, or merely required for the proof. In the latter case, many more BFT protocols could be used for checkpointing.

We have shown that our protocol is essentially a finality gadget, just as Afgjort [36], GRANDPA [37] and Gasper [46]. Through Figure 3.2, it we have shown how some of these protocols could be tweaked to enhance their functionality to the level of the protocol we have designed. Formally analyzing the security of GRANDPA, and tweaking Gasper to make it secure (with a formal proof) are interesting open problems that could be tackled using the tools of this chapter.

Finality gadgets/checkpointing could potentially offer many more properties. For example, they could protect against a dishonest mining majority [38]. They could potentially also be used to have lower latency, and even responsive confirmation of blocks. Designing a protocol that achieves these properties in addition to the ones we show is an exciting research problem. Designing an incentive-compatible finality gadget also remains an open problem. Finally, a system implementation of such protocols could lead to newer considerations, such as communication complexity or latency, which would pave the way for future research.

Chapter 4

Pricing and Routing in Payment Channel Networks

In this chapter, we turn our attention to *payment channel networks*, which is a tool to enhance a blockchain's transaction throughput. As discussed in Section 1.3.3, the longest-chain protocol has inherently poor throughput and consequently has high transaction fees. We also discussed how *layer-two blockchain mechanisms* overcome this limitation by allowing users to transact securely without recording their transactions on the blockchain. A payment channel network (PCN) is one such layer-two mechanism that is already being used in practice [15]. In recent years, there has been considerable research interest on PCNs, with a focus on improving their security as well as their efficiency. This chapter studies the long-term transaction processing efficiency of payment channel networks.

Motivated by the rich literature on communication networks [48, 49], we seek to understand PCNs better by studying a mathematical model of such a system. As the name suggests, a PCN is a network composed of multiple *payment channels*. A transaction flows through a path in the payment channel network in a manner similar to the flow of water in a network of pipes or the flow of bits in a communication network. Nodes in a PCN make transactions requests constantly over time. For each such request, the network decides whether to serve (i.e., execute) it or not, and if so, which path to route it along. Our mathematical model reveals that PCNs have fundamental capacity limitations, and, thus, they cannot serve transaction requests arbitrarily. This observation highlights the need for a principled protocol to handle transaction requests that arise in the network.

The main contribution of this work is a joint flow-control and routing protocol for PCNs, which we call the *DEBT (detailed Balance Transaction) control protocol*. The key ingredient of this protocol is the notion of *channel prices*, which is a price quoted by each channel to the network for routing transactions through it. The sum of channel prices along a path is interpreted as the price of the path. Based on these path prices, transacting pairs of nodes make flow-control and routing decisions for every transaction request that arises between them. Channels keep updating their prices over time based on the transactions that flow through them. Under stationary demand regimes, the protocol steers the prices and the transaction flows towards a stable, efficient operating point. We demonstrate this via mathematical analysis and simulations.

We begin this chapter by elaborating on payment channels and their basic operation in Section 4.1. We highlight that transactions in a payment channel must satisfy some basic feasibility conditions. This requirement translates into the following limitation for a payment channel: over

long periods of time, the net flow of money in one direction must equal the net flow of money in the opposite direction. Naturally, this constraint carries over to a PCN as well (Section 4.2): for transaction flows to be sustained indefinitely, the flows must satisfy *detailed balance* constraint. Because of this constraint, a PCN in general cannot serve every transaction request that arises from its users. Therefore, we focus on studying how a PCN can serve the maximum number of transactions over a long time, subject to the aforementioned constraint. We illustrate, via simple examples, that there are two different actions that can help achieve this objective: *flow-control* and *dynamic routing*. Flow-control refers to users regulating their rate of transactions by holding back on certain transactions as dictated by the protocol. Dynamic routing refers to a pair of transacting users switching between different paths over time in order to utilize the network's resources efficiently.

In Section 4.3, we present a discrete-time model of a payment channel network. The model specifies, among other things, the order of events in a time slot, the constraints imposed by the PCN, and the possible actions that transacting users can take. Section 4.4 is devoted to the design of the DEBT control protocol. We begin by posing the protocol's objective as solving the following constrained optimization problem: find the set of flows in the network that maximizes the fraction of transactions served while respecting the detailed balance constraint. Invoking convex optimization theory, we relax the constraint using Lagrange multipliers and obtain an equivalent, unconstrained form of the original problem. An iterative gradient-based algorithm to solve this unconstrained problem naturally follows. We then observe that this algorithm can be implemented as a network protocol. Interpreting the Lagrange multipliers associated with the dual problem as channel prices gives an intuitive understanding of the protocol. The protocol's convergence follows under some additional standard assumptions; this is presented in Section 4.5. Simulation results of the protocol are presented in Section 4.6 to illustrate the protocol's properties.

4.1 Payment Channels: Basic Principles

Construction and Basic Operation The idea of a payment channel was born at least as early as 2013 and has since undergone considerable evolution. We refer the interested reader to Section 3 of [11] and Chapter 5 of [50] for a historical survey of payment channels. Here, we describe the construction and operation of a payment channel at a level of generality that encompasses several designs. A payment channel is created by means of a special *funding transaction*, with two nodes depositing some money into a new two-node account (the channel). Money can be spent from this special account only when both nodes agree to do so. At the time of creation, both nodes also create and privately hold a *commitment transaction* which disburses the money held in the channel back to their personal accounts. This serves as a guarantee for each node that they can get their money back whenever they wish to.

Once a channel is created between two nodes, they can transact by exchanging messages between

themselves. Each message is simply a new commitment transaction, i.e., an agreement between the two nodes to split the escrowed fund in a certain portion between the two nodes. Thus, each transaction is reflected as a readjustment of the total value between the two nodes. None of these transactions need to be broadcast to the blockchain network. A channel is closed by recording one such commitment transaction on the blockchain, which effectively returns the appropriate balances to the two nodes' individual accounts. A payment channel also includes some inbuilt safety mechanisms by which an honest node can withdraw its funds even if the other node goes missing or acts maliciously.

One-directional Payment Channels The first payment channels invented were one-directional by nature; payments could only be made by one node to the other, but not the other way around. Such payment channels are analogous to gift-cards, as illustrated by the following example. Suppose a customer buys a gift-card for a certain coffee chain for a hundred dollars, paying for it with their credit card. This transaction gets recorded on the credit card's server. Subsequently, when the customer buys a coffee for five dollars each time using the gift-card, these transactions are solely between the store and the customer; in particular, they are not recorded by the credit card company or the customer's bank.

In the blockchain world, a customer may create a unidirectional payment channel by depositing some money into it and subsequently use this deposited money to make multiple small payments. Each payment consumes some of the escrowed amount, eventually depleting the entire fund, just as in a gift-card. At this stage, the payment channel must be closed and a new one opened if further transactions are to be made. The channel may also be closed before exhausting the entire amount, if either of the two nodes decide they do not wish to transact further.

Bidirectional Payment Channels The pioneering work of Decker and Wattenhofer [51] as well as Poon and Dryja [52] introduced bidirectional payment channels, i.e., channels that can support payments from either node to the other one. To visualize these payment channels, it is useful to think of a beaded wire, with the beads split among the two ends of the wire, similar to an abacus. Each bead represents a unit of money. Thus, the total number of beads is proportional to the total amount deposited, while the number of beads at each end denote the amounts owned by each node. A payment across a channel can be visualized as a movement of beads from the payer to the payee. From this point onward, the term payment channel is used to refer to a bidirectional payment channel.

Channel Capacity and Balances The total amount of money escrowed in a channel is called the *capacity* of the channel. The amount of money each node owns within the channel depicts the *balances* of the two nodes. While the capacity remains constant over time, the balances change with every transaction. The sum of the balances always equals the capacity. It is useful to think of the individual balances as the *channel's state*. In fact, a channel's state can be represented by

just one of the nodes' balances; the balance of the counter-node can be calculated by subtracting this number from the capacity.

Single-Shot Transaction Feasibility At any given point in time, a channel's balances impose a bound on the maximum value of a single transaction that can be made in either direction. To elaborate, consider a channel between two nodes A and B, and let their balances be x_A and x_B respectively. Then A cannot pay B more than x_A amount of money and B cannot pay A more than x_B amount of money in a single transaction. Usually, in a payment channel, transaction values are much smaller than the channel's capacity. Therefore, if the channel is fairly well-balanced, i.e., the balances are close to half the capacity, then the aforementioned feasibility considerations do not pose any restriction. However, if a channel's balances are skewed, certain transactions may be infeasible. For instance, if x_B is very small, it may not be feasible for B to pay A via the channel.

Flows Consider a scenario where a customer purchases two cups of coffee every day from a cafe, paying five dollars for every purchase. More so, the customer pays the cafe over a payment channel. This long-term transaction pattern can be represented as a *flow* of money across the channel. The *rate of flow*, in this case, is ten dollars per day. More generally, a flow through a channel is a concept that represents a steady transfer of money from one node to another over some considerable period of time. We can use the notion of a flow to represent a broader set of transaction patterns as well, such as a series of transactions whose values fluctuate randomly or periodically around some average value. In all cases, a flow represents a long-term sequence of transactions made at regular intervals.

A payment channel can support two flows in parallel, one in either direction. In a payment channel $A - B$, let $f_{A,B}$ be the flow rate from A to B and let $f_{B,A}$ be the flow rate in the opposite direction. The *total flow* through the channel is the sum of the flow rates, $f_{A,B} + f_{B,A}$. The *net flow* through the channel is the difference between the flow rates, $f_{A,B} - f_{B,A}$; here, we assume that each each channel has some implicit direction for computing the net flow. Throughout our discussion on flows, we assume there is some implicit unit of time to define flow rates.

The notion of flows also sheds some light on the choice of the term *channel capacity*. In the context of communication networks, this term denote the maximum rate of communication that the channel can support, i.e., the maximum number of bits (or information) that the channel can carry per unit time. For a payment channel too, there is a similar interpretation; its capacity (defined as the total escrowed fund) is proportional to the maximum possible total flow through the channel. Indeed, the total flow is maximized if each node sends its complete balance to the counter-node immediately after the previous transaction is completed. The resulting flow is proportional to the total escrowed fund. The precise flow rate depends on the duration of a time slot, which is implementation-specific. With some abuse of notation, we use the term capacity to denote the total escrowed fund, ignoring the time scale.

The Balance Condition for Flows In addition to a bound on the total flow rate, a payment channel also imposes the following *balance constraint*. A payment channel can sustain flows forever only if the net flow through the channel is zero. Put differently, a non-zero net flow cannot be sustained indefinitely. This fact is illustrated by the following example. Suppose, for instance, that on average, A pays B ten dollars per day and B pays A five dollars per day. When this happens, A's balance in the channel decreases continuously. Whatever be the channel's capacity, eventually A's balance gets completely exhausted. When this happens, the channel will not be able to support further payments from A to B.

In such a scenario, A could wait for B to make some payments to A. These transactions would replenish some balance on A's side. However, this replenishment occurs at a rate of five dollars per day. Since A's payments are ten dollars each day, A will only be able to pay B once in two days. Effectively, the flow rate from A to B drops by half to five dollars per day. At this stage, the channel has zero net flow. The above example illustrates how a channel naturally enforces the balance constraint on flows through it. As this example shows, it is helpful to distinguish between the *demand* (or request) for flows and the flows *served*.

On-Chain Rebalancing The above example assumes that the demand for flows are *elastic*, i.e., A is willing to tolerate a smaller flow rate (five dollars per day) than its original demand (ten dollars per day). However, if flows are *inelastic*, i.e., the entire demand must be served through the channel, then the channel can reset its balances via an on-chain transaction. With periodic *on-chain rebalancing*, a channel can support imbalanced flows (non-zero net flows).

On-chain rebalancing can be done in one of the following ways. One option is that A pays B some amount via an on-chain transaction, with B paying A back the same amount on the channel. If this amount is half the channel capacity, the entire operation would rebalance the channel. Alternatively, the nodes may open a new, perfectly balanced channel between themselves (and possibly close their older channel). Either of these methods incur the transaction cost of at least one on-chain transaction.

4.2 Long-Term Operations in Payment Channel Networks

Having sketched out the basic properties of a payment channel, we now turn our attention to the main object of interest in our work, namely, payment channel networks. The concept of a payment channel network was born with the design of the Bitcoin Lightning Network [52]. Such a network is formed among a collection of nodes with bidirectional payment channels among them. These nodes coordinate with each other to support multi-hop transactions through their channels. Thus, nodes that do not share a channel can transact by routing the payment through a path of channels. Intermediary nodes receive money from one channel and send money along a different channel.

Creating a network of inter-operable payment channels allows nodes to transact with a large

number of others without creating a channel with every one of them. This saves costs on two fronts: the large transaction fee spent on opening and closing a channel and the opportunity cost of locking up funds in channels. A payment channel network allows users to make transactions with high volume and with very low transaction fees. It is particularly useful for micro-transactions, i.e., transactions whose value is much smaller than the blockchain transaction fees itself.

4.2.1 Routing

One of the most challenging aspects of executing transactions in a payment channel network is the task of *routing*, i.e., finding a path in the network along which the transaction can be sent from its source to its destination. Broadly speaking, there are two classes of routing paradigms: *source routing* and *hop-by-hop routing* (also called as local routing [11]). In source routing, the source specifies the entire path from itself to the destination. The path information is encoded in the messages that accompany any transaction; intermediate nodes simply follow the prescribed path. Source routing requires nodes to know the topology of the network. In hop-by-hop routing, each node simply passes the transaction on to an appropriate neighbor; this neighbor then forwards the transaction on as it deems fit. This process continues until the transaction reaches its destination. This style of routing does not require knowledge of the complete network topology, and may be beneficial in scenarios with rapidly varying network topology.

In this work, we assume that all transactions are source routed. Further, we assume that each pair of transacting nodes have a fixed set of paths between them. Every transaction between a given source-destination pair is routed along one of these paths. These paths could be arbitrary; we do not delve into the graph-based algorithms (such as shortest-path algorithms) that are used to find such paths. In practice, new channels are added and old channels removed from the network. However, this happens on a slower time scale. If a channel is removed, nodes simply discard all paths that use that channel from their consideration. When new channels are added and other nodes become aware of it, they may bring new paths that use these channels into their purview. In this work, we assume that the topology of the network remains unchanged over time. Thus, the issue of updating the paths does not arise.

The rationale behind these assumptions is as follows. Firstly, the popular Bitcoin Lightning Network implements source routing rather than hop-by-hop routing; thus, our assumptions reflect this system better. Secondly, the topology of the system is usually public knowledge, so it is not unrealistic to assume that nodes have enough knowledge to compute paths among themselves. Thirdly, the topology changes on a much slower time scale than the rate at which transactions take place. Moreover, our paradigm of fixing a small set of paths and switching among them captures a variety of routing schemes.

Just as in a single channel, feasibility considerations come into play while routing transactions through the network. For a transaction to be feasible along a path, the balance of all channels in the path (on the appropriate side) must be at least as large as the transaction value. One of the

most basic goals of any routing protocol is to find a feasible path for every transaction. If nodes do not have access to the state information of the network (i.e., the channels' balances), it is possible that they might choose a path that is infeasible. Without any state information, routing essentially boils down to trial and error: try a certain path, if infeasible, try a different path, and repeat until successful or all paths have been exhausted. Thus, it is imperative to have some state information for routing efficiently.

Even with state information, a routing protocol may not always be successful in finding a feasible path. Indeed, if all channels leading out of a node have no balance left on the source's side, then no feasible path exists for transactions made by this node. Even in a less extreme scenario, if the transactions are of large value, a feasible path may not exist. Another possibility is that multiple transactions at the same time clog a single channel. Although each transaction by itself is feasible, all of them together are infeasible. Last but not the least, if nodes have partial, inaccurate, or delayed state information, any routing protocol is likely to choose infeasible paths. When an infeasible path is chosen, the network immediately drops the transaction.

4.2.2 Detailed Balance Condition For Flows

We begin with extending the notion of flows, introduced in Section 4.1, to payment channel networks. As before, a flow represents a steady movement of funds from a payer (source node) to a payee (destination node) over some considerable period of time. However, in the context of a payment channel network, a flow is also associated with a fixed path. Thus, we speak of a flow of rate f_p along a path p in the network. As we are interested in studying the long-term transaction processing efficiency of PCNs, our description of transactions will largely be in terms of flows.

In certain scenarios, it is possible that a given pair of nodes transact over multiple paths. In this case, it is appropriate to associate a flow with a multi-path. To elaborate, a multi-path flow is a collection of single-path flows, $(f_1, \ldots, f_k)$, each of which represent transactions executed over a particular path from the same source to the same destination. In some PCNs, a single transaction can be split along multiple paths [53]. However, even if this is not the case, a multi-path flow can represent a scenario where a pair of nodes rotate their routing choices among different paths such that each path carries a steady flow on the average.

So far, we have considered flows from one source node to one destination node. However, in a PCN, many nodes transact in parallel. Therefore, we are led to consider scenarios where multiple flows move through the network simultaneously. Thus, the set of all flows in the network can be described by a collection of multi-path flows, with each multi-path flow representing transactions between a unique source-destination pair. As mentioned above, each multi-path flow is further composed of some single-path flows. At times, it is more suitable to represent the flows as simply a collection of single-path flows, treating flows among the same source-destination pair independently. With this viewpoint, we can represent all flows in a PCN by a vector f, indexed by the set of all paths carrying flows in the network. As mentioned in Section 4.2.1, we only consider a subset of

all possible paths for routing.

Next, we turn our attention to the cumulative effect of multiple flows on an individual channel. Just like currents in an electrical network or water in a network of pipes, transaction flows in a PCN add up to form a combined flow through each payment channel. The flow in a channel, along one particular direction, is the sum of all flows along paths that traverse the channel in that direction. The flow in the opposite direction can be computed likewise. The total flow in a channel is the sum of these quantities, while the net flow is their difference. These notions naturally carry over from the case of a single channel. The net flow (or the total flow) can be represented as a vector $\tilde{f}$, indexed by the channels in the network.

As we saw in Section 4.1, a non-zero net flow through a channel skews the channels balances and thus cannot be sustained indefinitely. Thus, for a payment channel network to be able to serve transactions over a long period of time, the flows must be routed such that the net flow on each channel is zero. We call this condition the *detailed balance constraint*; flows that obey this condition are called *detailed balance flows*. Some small fluctuations over time can be accommodated; in other words, over a short time frame, it is possible that there is a non-zero net flow across a channel. However, over moderately long time frames, the total flow in one direction must equal the total flow in the opposite direction for every channel.

The detailed balance condition has an important consequence: in general, a payment channel network cannot serve arbitrary transaction requests for a long period of time (assuming they do not perform periodic on-chain rebalancing). Indeed, we observed this for the trivial case of a single payment channel in Section 4.1: if the flow rate requested (or demanded) in the two opposite directions of a payment channel were unequal, then the channel could not serve the entire demand. In the case of a single channel, the only option available was to drop transaction occasionally from the flow with the higher demand such that the flow rates in the two directions matched. The demand not served is thus equal to the difference of the requested flow rates. In a network, routing smartly can maximize the fraction of demand served while maintaining the detailed-balance condition. However, in general, it cannot serve the complete demand. In the following subsection, we characterize the conditions under which a PCN can serve all the entire demand.

4.2.3 Detailed Balance Flows and Circulations

For a PCN, a *circulation* is a demand pattern which satisfies the following condition: the total incoming demand is equal to the total outgoing demand for every node in the network. For example, in a network with three nodes, A, B, and C, if A wants to pay B , B wants to pay C, C wants to pay A, and all of them want to transact at a rate of ten dollars per day, the demands form a circulation. However, if the demand from A to B is raised to twenty dollars per day while the other demands remain the same, this demand pattern is not a circulation. Note that a circulation imposes a constraint on flows at each node, whereas the detailed-balance condition imposes a constraint at each edge. The following result establishes the equivalence between circulations and

detailed-balance flows.

Proposition 4.1. *Consider a payment channel network whose underlying graph topology is connected, i.e., there is at least one path from every node to every other node. Let each pair of nodes in the network have some fixed demand for transaction flows, possibly zero. Then, the entire demand can be served by detailed balance flows if and only if the demands form a circulation.*

Proof. By definition, if flows satisfy the detailed balance condition, the net flow on each channel must be zero. As a consequence, for each node, the net flow into the node is equal to the net flow out of the node. This can be seen by fixing a particular node and adding up the total incoming and outgoing flows along each of the channels incident on it. Since these flows satisfy the complete demand, the demands must be a circulation.

To prove the converse, suppose we know that the demands form a circulation. Since the network is connected, there is at least one spanning tree in the network. Consider any fixed spanning tree; such a tree contains a unique path from every node to every other node. Assume that the entire demand is routed along the paths of this spanning tree. Then, each channel in the tree carries a zero net-flow. This can be argued as follows. Consider any one edge in the spanning tree. It partitions the set of nodes in the network into two, based on which side of the edge they lie on. Since the net flow into each of these two sets of nodes is zero (because the demands form a circulation and the entire demand is being served), the edge in question must carry a zero net flow. This argument holds for each edge in the spanning tree. Since other channels do not carry any flow, the flows in the entire network satisfy the detailed balance condition. Thus, we have shown that there exists a way to serve the entire demand via detailed balance flows. □

Note that for a circulation demand to be successfully served entirely by a routing scheme, there are a couple of other conditions that must be met. For one, the channel capacities must be large enough to carry the total flow requested through it. Second, the above result only shows the existence of a set of paths that can serve the entire demand; finding such a set of paths in a decentralized fashion is a different problem altogether. At the very least, the routing protocol must operate with a sufficiently diverse set of paths to work with; otherwise, it is possible that no spanning tree exists among the paths considered. The following section highlights the importance of considering multiple paths and illustrates a decentralized state-dependent protocol to route transactions optimally.

4.2.4 The Benefits of State-Dependent Routing

To illustrate the benefits of state-dependent routing, let us consider the following example where the demands form a circulation. As per Proposition 4.1, there should exist a routing scheme to satisfy the complete demand in this case. We illustrate how state-dependent routing discovers such a routing scheme.

Example 4.1 (Circulation)**.** *The payment channel network consisting of three nodes A, B, and C. Each pair of nodes have a channel between them. Each channel containing a balance of one hundred coins with each node (the capacity of each channel is two hundred). The demand for flows is as follows:*

- *A wants to pay B ten coins per unit time.*
- *B wants to pay C ten coins per unit time.*
- *C wants to pay A ten coins per unit time.*

For Example 4.1, suppose each source routes payments to its chosen destination only via the shortest path, i.e., via the link directly connecting the two nodes. Then, after 10 units of time, all channels will clearly get imbalanced. All the balance in the $A - B$ channel will be on B's side, the balance on the $B - C$ channel will be on C's side, and the balance on the $C - A$ channel will be on A's side. With the channels being imbalanced in such a manner, the nodes cannot continue transacting along these paths. However, a careful look reveals that even in this imbalanced state, the nodes could continue transacting by routing their transactions via the longer path. E.g., A can pay B by routing a transaction along the path $A \to C \to B$. Likewise, the two other transacting pairs, $B \to C$ and $C \to A$, can also route their transactions via a longer path.

In this example, routing along the longer path once rebalances the channels to some extent, allowing transactions to flow along the shortest path again twice. Thus, the requested flows can be served perennially, provided each transacting pair chooses the longer route once every three times they transact; in other words, choose the shorter path twice as often as the longer path. Effectively, each transacting pair splits its flow along the shorter and longer path in the ratio $2 : 1$. Considering the flows among all transacting nodes together, we see that this arrangement leads to detailed balance flows. The example underscores the importance of each source-destination pair considering multiple paths for routing transactions and switching among them dynamically.

In a larger network, how can transacting node pairs discover a suitable routing scheme in a decentralized manner? To address this challenge, Varma and Maguluri [16] propose a decentralized, state-dependent routing protocol for a payment channel network. In their protocol, channels quote prices for routing a unit of flow through in either direction. The price of a channel in any given direction could be positive or negative. Positive prices discourage further flows through the channel in that direction, while negative prices encourage more flow. The price for a path is the sum of prices on the channels in the path. A node routes its transactions along the path with the minimum price. Each edge updates its prices based on the net flow through it. The protocol is decentralized in the sense that each transacting node-pair makes its routing decisions without coordinating with the others, and each edge updates its prices based only on the flow through it. Moreover, the channels do not need to know the demand patterns ahead of time. The term state-dependent reflects the fact that the routing decisions are made based on the prices, which in turn depends on the state of the system. The protocol is shown to be throughput-optimal [16]. This means that

as long as the demands form a circulation, they can be routed such that the corresponding flows satisfy detailed balance and thus, be supported perennially.

The routing scheme we described in the beginning of this section can be viewed as a rudimentary decentralized, state-dependent protocol. Here, the state refers to the channel's balances, in particular, whether a path is feasible or not. Nodes would naturally have a preference for the shorter path, but would switch to the longer path if the shorter path was infeasible. In contrast, a state-independent scheme would be one where nodes always route one in every three transactions along the longer path, or split each transactions along the shorter and longer path in the ratio $2:1$. A centralized scheme would be one where A, B, and C coordinate their routing decisions, with $C \to A$ transactions always being routed via the path $C \to B \to A$, while $A \to B$ and $B \to C$ route along the shortest path. Any of these schemes would maintain detailed balance flows in the given example. However, the protocol in [16] has the advantage of being decentralized, robust to random perturbations in the demand, implementable on arbitrary network topologies with arbitrary demands, and provably optimal as long as demands are circulations.

4.2.5 Deadlocks: Motivation for Flow-Control

The protocol proposed by Varma and Maguluri [16] always routes every transaction request along some path or the other. When the demands form a circulation, it is able to serve the entire demand. What happens when the demands are not a circulation? We explore this question with the help of the following two examples.

Example 4.2. *The payment channel network consisting of three nodes A, B, and C. Each pair of nodes have a channel between them. Each channel containing a balance of one hundred coins with each node (the capacity of each channel is two hundred). The demand for flows is as follows:*

- *A wants to pay B twenty coins per unit time.*
- *B wants to pay C ten coins per unit time.*
- *C wants to pay A ten coins per unit time.*

Example 4.2 is a small modification of Example 4.1, where the rate of flow requested from A to B is higher. Thus, the demands no longer form a circulation.

Suppose that, initially, all transaction requests are served. Then, node A receives coins at a rate of ten coins per unit time, while it spends at a higher rate. Therefore, it will eventually run out of coins, whatever routing strategy the network adopts. To elaborate, the network will eventually reach a stage where both the channels $A-B$ and $A-C$ will have no balance left towards A's side. At this stage, it would be infeasible to route transactions further from A to B along either of the two paths, forcing $A \to B$ transactions to be dropped.

Meanwhile, transactions from B to C and from C to A continue to flow. These transactions will eventually replenish balances on the two imbalanced channels, allowing A to resume transacting

with B. However, the net flow from A to B would be limited to ten coins per unit time; once again, this can be argued on the basis of the net flow at node A. In other words, at least half of $A \to B$ transactions would have to be dropped, irrespective of the routing strategy. Note that the net flow of transactions served is identical to the flows in the Example 4.1.

In general, whenever the demands do not form a circulation, routing every transaction request is bound to lead to flows that do not satisfy detailed balance, and are therefore bound to lead some channels to get imbalanced. Because of these, some paths can no longer route transactions. Transactions that are attempted to be routed along any path that use the imbalanced links are usually dropped. Sometimes, there are flows in the opposite direction that replenish the balances along these links. In such a case, transactions can continue to flow through this channel, but at a reduced rate, so as to match the rate of replenishment. We saw this in the case of a single channel in Section 4.1 as well as in the above example.

In some scenarios, it is possible that no flow in the PCN can replenish the balances of the channel, which mean the channel remains imbalanced forever. In this case, all flows using that path are permanently stalled. It is possible that some of these flows could have been part of a detailed balance flow had the channel not been imbalanced. This phenomenon is called a *deadlock*. When the network goes into a deadlock, the total flow served in steady state is less than the what can potentially be served if channels were perfectly balanced. Deadlocks were first identified by Sivaraman et al. [54] and studied in more detail in a follow-up paper [17]. We now illustrate the phenomenon of deadlocks with the following example [17].

Example 4.3 (Deadlock)**.** *The PCN consisting of three nodes A, B, and C. It has two channels, $A - B$ and $B - C$. Each of the channels has a balance of a hundred coins on each side. The demands for flows are as follows:*

- *$A \to C$ at a rate of ten coins per unit time*
- *$C \to A$ at a rate of ten coins per unit time*
- *$B \to A$ at a rate of ten coins per unit time*
- *$B \to C$ at a rate of ten coins per unit time*

In Example 4.3, there is only one path between every pair of nodes, so the question of routing does not arise. The only decision each node needs to make is whether to actually route a transaction request through the network, or drop it (equivalently, execute it by some other means). From the demand patterns, we see that only the transactions between A and C can be supported in the long run without rebalancing, since these flows would leave the channel balances untouched. The transactions from B must eventually stop because they are not matched by transactions flowing in the opposite direction.

Suppose, initially, all transactions are served by the network. After some time, both the channels get imbalanced, with all the balance being on the side of A and C and no balance left on B's side.

At this stage, the transactions from B are forced to stop because it is infeasible for the network to route them. However, note that it is also infeasible for the network to route transactions from $A \to C$ and $C \to A$. For an $A \to C$ transaction to be feasible, there must be sufficient balance on both channels: in channel $A - B$, on the side of A, and in channel $B - C$, on the side of B. However, the $B - C$ channel is imbalanced in the wrong direction; thus, it is unable to carry the $A \to C$ transaction. Similarly, $C \to A$ transaction also cannot be served by the network. Thus, the imbalance in some channels blocks out transactions that could otherwise be served in steady-state. Moreover, the network is stuck in this scenario, unless some channels rebalance externally. This phenomenon is called a deadlock [54, 17].

Note the difference between Examples 4.2 and 4.3. Both examples have demands that cannot be sustained in stead-state. In Example 4.2, nodes could continue sending transactions through one path or the other until it was infeasible to send through both; however, this 'greedy' action of the nodes did result in optimal long-term throughput. In contrast, in Example 4.3, we see that greedy actions led to zero steady-state throughput, which is sub-optimal. Note that the deadlock was caused because of the imbalanced channels, which in turn was caused by the transactions from B to C and A.

Now suppose B held back its transactions, even when the channels had sufficient balance to route its transactions. In other words, the flows from B to A and B to C are preemptively stifled. In such a scenario, the flows from A to C and C to A become feasible, and these can be sustained forever. This would lead to the optimal steady-state throughput. Note that stifling the flows from B does not reduce the long-term throughput because these flows could not have been supported by the network anyway. This action, where transaction requests among nodes are not passed on to the network, despite them being feasible, is called as *flow-control*.

Can flow-control be used to avoid deadlocks in arbitrary payment channel networks? The main challenge is to stifle only those flows that remain unbalanced, allowing the rest to pass through. Moreover, this selective flow control must be done in a decentralized fashion. In Section 4.4, we show that the routing protocol proposed by Varma and Maguluri [16] can be extended to perform flow-control as well. In particular, a transacting node-pair decides the rate of flow based on the price of the cheapest path between themselves. If the cheapest price itself is too high, they do not request transactions to be carried out over the network.

4.2.6 Possible Objectives for a PCN Protocol

Based on the examples above, we conclude that, for arbitrary transaction arrival processes, a PCN cannot serve all transaction requests without periodically performing on-chain rebalancing. As on-chain re-balancing is an expensive operation, the network is incentivized to reduce the extent of this action as much as possible. If channels are to avoid doing so, they must decline to serve certain transaction requests. Thus, there is a trade-off between maximizing the rate of transactions served and minimizing the rate of on-chain transactions. There are two possible objectives that

the network could try to achieve:

- Minimize the rate of on-chain rebalancing transactions while routing all transaction requests.
- Maximize the rate of transactions served on the network while ensuring there are no on-chain rebalancing transactions.

Here, the word rate of transactions refer to the total monetary value of the transactions involved per unit time. In this work, we focus on the latter design criterion. In other words, we aim to design a network protocol that serves only those transactions that can be routed without requiring on-chain rebalancing. Based on the examples seen in this section, we see that such a protocol must perform both routing and flow-control. Although the importance of routing is clearly illustrated in many works [11], the notion of flow-control is new in payment channel networks. Indeed, on the surface, it seems unreasonable for transacting nodes to hold back their transactions even if it is feasible to serve them. However, as we show in the discussion above, flow control is beneficial in avoiding deadlocks.

Finally, note that a transaction that is not served on the payment channel network can be served in various other ways. For example, nodes could record their transaction directly on the blockchain, or they could create a new channel between them and route the transaction through it. The network protocol we present naturally allows for different possibilities on how to serve the transactions dropped by the network. Arguably, a more holistic design of the network behavior would involve minimizing the total cost of serving all transaction requests, allowing for the aforementioned options of serving a transaction. However, factoring in the cost of these options is nontrivial. For example, the cost of opening a new transaction would depend on the capacity of the channel, which in turn is dictated by the opportunity cost of keeping funds locked in a channel.

4.3 A Discrete-Time Model of a PCN

In this section, we present a mathematical model of a payment channel network. This model serves as the bedrock for the routing and flow-control protocol that we present in Section 4.4. Our model incorporates many important ideas from the models proposed by Varma and Maguluri [16] and Sivaraman et al. [17]. The model described here is not intricately tied to the protocol we propose next. In fact, it can be used in future work to study the behavior of PCNs under different modeling assumptions and other protocols. In Section 4.5, we make additional assumptions on the model in order to prove convergence.

The model presented here includes a description of the arrival of transaction requests to the PCN. We assume these transaction requests are determined by factors external to the system. Given a transaction request, the corresponding source and destination node need to decide whether or not to have it served by the network. This distinction between transaction requests and transactions served is crucial in understanding payment channel networks. Lastly, we typically assume that

these transaction requests form a stationary process. Focusing on the stationary regime allows us to meaningfully define the notion of an optimal flow, as well as convergence of a protocol to this optimal flow. However, this assumption is not necessary to describe a mathematical model of a PCN.

We model the PCN as a discrete-time system. In each time slot, the following actions take place in the given order:

1. Transaction requests arise among nodes in the network. Each transaction is has a source, a destination, and a value. Multiple transaction requests can arise in the same slot. The arrival of transactions to the network is controlled by exogenous factors, independent of the happenings in the network; we treat the process as given.

2. The nodes make routing and flow-control decisions. For every transaction request, the corresponding source-destination pair decides what fraction of the amount to route, and along which path (or multi-path). We assume the nodes have access to some information regarding the state of the system; the precise information used is protocol dependent.

3. The payment channel network serves the transaction requests. If a channel cannot serve the requested transactions by virtue of being imbalanced, it rebalances itself. We assume conditions are such that all transaction requests that are passed on from the nodes to the network are actually served. After the transactions are served, channel balances are updated.

4.3.1 Key Components of the System

The PCN as a graph The network consists of a set of nodes V and a set of channels E between pairs of nodes. The nodes are numbered $1, 2, \ldots, |V|$. A channel connecting nodes u and v is denoted (u, v). We use the convention that the lower index vertex is written first; e.g., in the channel (u, v), $u < v$. Each channel has a certain capacity, which refers to the total amount of money escrowed in the channel. Let $c_{u,v}$ denote the capacity of channel (u, v) and let $c \in \mathbb{R}_+^E$ be a vector denoting the capacities of all the channels in the network. Thus, the tuple (V, E, c) specifies a weighted, undirected graph with numbered nodes. We assume this graph remains unchanged throughout the period of operation.

Paths Each transaction request that is passed on to the network needs to be routed from its source to its destination along a path in the PCN. A path may have cycles, but we assume that it traverses each edge at most once. Any such path can be represented by a vector $r \in \{-1, 0, 1\}^E$ as follows:

- let $r_{u,v} = 1$ if the path traverses the channel (u, v) in the direction $u \to v$,
- let $r_{u,v} = -1$ if the path traverses the channel (u, v) in the direction $u \leftarrow v$, and

- let $r_{u,v} = 0$ if the path does not traverse the channel (u, v) in either direction.

In any graph, there could be multiple paths from a source node i to a destination node j. We assume that every node pair has a fixed set of paths between them that they consider for routing transactions; this set need not include all possible paths between i and j. Denote these paths by $p_{i,j,1}, \ldots, p_{i,j,k}$, and the set of such paths by $P_{i,j}$. Let P denote the set of all paths, $\cup_{i,j} P_{i,j}$. Let R denote the routing matrix indexed by $P \times E$ with entries in $\{-1, 0, 1\}$, where each row corresponds to a particular path, and each column corresponds to a particular channel.

Balances The term *balance* refers to the distribution of the escrowed fund in a channel between the two nodes involved. At the beginning of slot t, let the balance of node u in channel (u, v) be $x_{u,v}[t]$. It follows that the balance of v in the same channel is $c_{u,v} - x_{u,v}[t]$. Let $x[t] \in \mathbb{R}_+^E$ denote the vector of balances. By convention, we always denote the balance of the lower-indexed node of each channel in x, and the balance of the opposite end is inferred from the capacity vector. The balance vector always satisfies $0 \leq x[t] \leq c$, where the vector inequalities are to be interpreted as the inequalities holding component-wise. The vector $x[t]$ denotes the state of the PCN at time t.

4.3.2 Transaction Requests

In every slot t, multiple transaction requests arrive, with arbitrary sources, destinations, and amounts. In general, there could be multiple transaction requests from source i to destination j in slot t; let $A_{i,j}[t]$ denote the set of such transactions. Note that $A_{i,j}[t]$ is distinct from $A_{j,i}[t]$. In particular, we shall frequently consider one of the two following cases:

- **Atomic:** $A_{i,j}[t]$ consists of one transaction of value $a_{i,j}[t]$, which must be processed as a whole (either served, queued, or dropped).
- **Infinitely divisible:** $A_{i,j}[t]$ consists of infinitely many infinitesimal transactions of total value $a_{i,j}[t]$. Any fraction of this value can be served, with the rest being dropped. We treat this as a single transaction request of amount $a_{i,j}[t]$, but allows for fractional amounts to be served.

The atomic regime and infinitely divisible regime denote two extremes; in general, there are a finite number of transaction requests, of which any subset can be served depending upon the conditions prevailing.

Demand Distribution Throughout this work, our focus will be on stationary demand regimes. To be more specific, for any pair of nodes (i, j), we assume that the demands as a function of time are independent and identically distributed (i.i.d.). In particular, this means that $(a_{i,j}[t] : t \in \mathbb{N})$. is an i.i.d. process. Moreover, the demand processes across different nodes are also independent. A special case here is the regime of *constant demands*, where $a_{i,j}[t] = a_{i,j} \; \forall \; t, \; \forall (i, j)$. The vector

$a = (a_{i,j})_{(i,j)\in V\times V}$ is called the demand vector. Note that the stationarity assumption allows for both atomic as well as infinitely divisible transactions.

4.3.3 Flows

Once a pair of nodes decide to have a transaction served by the network, they also specify one or more paths for the transaction to flow along. Thus, each transaction that is served is associated with a certain amount and a path (or multi-path). Define a flow to be a transaction amount coupled with a path. Since there are many transactions being requested and served in a single time slot, there are multiple flows in parallel. The set of flows in slot t is represented as a vector $f[t] \in \mathbb{R}_+^P$. The components of this vector, denoted by $f_{i,j,k}[t]$, represent the amount of money sent along path $p_{i,j,k}$ in slot t. The total flow from node i to node j along all possible paths is denoted by $f_{i,j}[t]$. Thus,

$$f_{i,j}[t] \triangleq \sum_{k:p_{i,j,k}\in P_{i,j}} f_{i,j,k}[t]$$

We clarify on the usage of the term flow. Here, we have used this term to denote the one particular transaction (in one particular time slot) coupled with one particular path. We denote such flows by $f[t]$, explicitly mentioning the time slots. Earlier, in Section 4.2, the term flow had the interpretation of a steady movement over transactions along a path over a long period of time. In this case, we use the notation f (without t) to denote the vector of long-term flow rates. The precise meaning of the term will be made clear from the context. Naturally, the two concepts are related. Under stationary demand regimes, we expect instantaneous flows $f[t]$ to converge to (or oscillate around) some fixed flow rates f.

Feasible Flows In any slot t, nodes request a set of flows $f[t]$ to be routed by the network. It is possible that it is infeasible for some channels to route the requested flows through them due to insufficient balance. A set of flows can be served entirely if and only if there is sufficient balance on each side of each edge to route all the flows through in that direction simultaneously. Given a balance vector $x[t]$, a flow vector $f[t]$ is feasible if:

$$R^+ f[t] \leq x[t]; \quad R^- f[t] \leq c - x[t].$$

Here, R^+ is the matrix obtained by turning all -1s to 0s in R and R^- is the matrix obtained by turning all 1s to 0s and -1s to 1s in R. Thus, $R = R^+ - R^-$. Suppose a feasible flow vector has been requested. Then, after the flow is routed, the balances are updated as:

$$x[t+1] = x[t] - Rf[t].$$

A feasible flow vector ensures that $x[t+1]$ satisfies $0 \leq x[t] \leq c$.

Next, we elaborate on how the network handles scenarios where the requested flows are infeasible.

On-Chain Rebalancing When a certain flow is infeasible, there are one of two possibilities. One possibility is that some channels have skewed balances, and are therefore unable to serve the flows. The other possibility is that the flow rates requested exceed the channel's capacity. Of course, it is possible to have both scenarios as well. We elaborate on both of these cases, and describe what the channel does in either of the scenarios.

- One possibility is that the channel (u, v) is unable to route flows $f[t]$ because its balances are highly skewed. However, if the channel was evenly ($x[t]_{u,v} = c_{u,v}/2$), then it would be able to satisfy the flows requested. This means that either $(R^+ f[t])_{u,v} > x[t]_{u,v}$ or $(R^- f[t])_{u,v} > x[t]_{u,v}$, but not both. Let us assume, without loss of generality, that it is the former. In such a case, channels reset themselves with node u paying node v an amount of $c_{u,v}/2 - x[t]_{u,v}$ via an on-chain transaction. node v then transfers the same amount to u via the channel, which brings back parity in the balances. Having performed the balancing operation, channels route the flows requested. We assume that all of these operations happen within one slot.
- A second possibility is that the channel channel (u, v) is not able to route the requested flows $f[t]$ because the total flow requested exceeds its capacity. Mathematically, this means $(R^+ f[t])_{u,v} + (R^- f[t])_{u,v} > c_{u,v}$. Such a flow vector would be infeasible to route, irrespective of the channel balances. In such a scenario, channels do not rebalance. Instead, each channel drops some subset of flows being routed through it such that the remainder can be routed.

In the following sections, we assume that channel capacities are sufficiently large such that if channels were indeed evenly balanced, they will be able to route arbitrary transaction requests. Thus, the second of the two scenarios listed above does not arise. We discuss this point further in Section 4.3.4.

Detailed Balance Flows In Section 4.2, we discussed the notion of detailed balance flows. Mathematically speaking, a flow vector f is said to satisfy the detailed balance condition (equivalently, said to be a detailed balance flow) if

$$Rf = 0.$$

For any flow vector f, the term $(Rf)_{u,v}$ represents the net flow in the direction $u \to v$ along the channel (u, v); a negative value would mean that the net flow is in the direction $u \leftarrow v$. A detailed balance flow is such that the amount of money flowing through each channel is equal in the two opposite directions. Thus, after a detailed balance flow has been served, the balances on all the edges remain the same as before. Note that for a detailed balance flow to be routed, the flow must also satisfy the feasibility conditions noted above. The set of feasible, detailed balanced flows is maximal when all edges are perfectly balanced ($x = c/2$).

4.3.4 Large Capacity Regime

In the protocol presented below, it will be useful to consider a regime where the channel capacities are, roughly speaking, sufficiently large compared to the transaction values. Large capacities allow the protocol some room to steer the network's operating point from an undesirable state to a desirable one. Below, we make this notion precise.

Consider a payment channel network where some pairs of nodes wish to transact. This set of transacting nodes may be arbitrary, but is typically much smaller than the set of all pairs nodes. Among these transacting nodes, let there be a fixed set of paths along which transactions may possibly flow. Here too, the set of paths may be arbitrary, but is typically much smaller than the set of all possible paths between a given source-destination pair. Further, assume that any transaction that arrives to the network is bounded above by a constant B. Lastly, assume that all the channels are perfectly balanced to begin with.

Suppose that each transacting pair of nodes receives a transaction of value B. Let F be the set of flow vectors obtained by considering every possible routing assignment of paths to transactions. The PCN is said to have *large capacity with respect to* B if all flow vectors $f \in F$ are feasible. Note that it suffices to consider transactions of value exactly B in the above definition, as it ensures feasibility of any lower valued transactions. The above notion also makes sense in scenarios where transaction values are random, and may exceed some reasonable bound B very rarely; in this case, the network can decide to not serve any transaction of value exceeding B, thereby retaining the ability to serve most of the transactions. When we make the statement that a given PCN is operating in the large capacity regime (without reference to a specific value B), we mean that the transaction values are bounded above by some appropriate constant, much smaller than the capacities of the channels.

A stronger notion of the large-capacity regime can be defined as follows. Let $k \in \mathbb{N}$ be a parameter. Then, the PCN is said to have *k-fold large capacity* (with respect to B) if every sequence of k flows $(f_1, f_2, \ldots, f_k : f_i \in F \ \forall \ i)$ is feasible, starting from the perfectly balanced state. For $k = 1$, we get back the original definition.

4.4 A DEBT (DEtailed Balance Transactions) Control Protocol for PCNs

The model described in the previous section provides a framework for discussing network protocols for PCNs. In this section, we present a joint flow-control and routing protocol that can be implemented on such networks. The goal of the protocol is the following: under stationary demand conditions, the protocol should serve as large a portion of the transaction requests as possible, while ensuring that the flows satisfy the detailed balance condition. We call this protocol the DEBT (DEtailed Balance Transactions) control protocol. Thus, the protocol seeks to serve as large a fraction of transactions as possible without incurring the high cost of periodic on-chain rebalancing. We refer the reader to Section 4.2 for the motivation behind this protocol.

The key idea of the protocol is that channels quote prices to nodes for routing flows through them. The price of a path is the sum of the prices of the channels along the path. Nodes choose their flows depending on the path prices. A large path price signals the nodes to reduce the flow along the path, while a low path price provides an incentive to increase the corresponding flow. If all path prices between a pair of nodes is large, they hold back on their transactions. The channel prices are adjusted in a manner such that eventually, only a detailed balance flow circulates through the network, while all other flows are stifled off. Crucially, the protocol allows channel prices to be negative as well.

The DEBT control protocol is an iterative one. Initially, all channel prices are zero; consequently, the path prices are zero as well. Therefore, all transaction requests are served and routing choices are made arbitrarily. As flows move through the network, channels adjust their prices based on the net flow through them. For a channel (u, v), a net flow in the direction $u \to v$ leads to an increase in the price for further flow in that direction. Thus, it dissuades further flow in that direction. The price in the opposite direction is simply the negative of the forward price. Imbalanced flows eventually cause channels prices along their paths to become large enough that they get stifled. However, balanced flows continue to see moderate path prices (possibly negative), and, therefore, continue to flow.

During transient periods, it is possible that flows along channels are not balanced; indeed, without an imbalanced flow, prices would not move at all. These transient imbalanced flows skew the channels' balances. If the channel capacities are large enough, they will not have to undergo any rebalancing. If they are more moderate in value, some channels may undergo rebalancing during the transient phase. However, the rate of rebalancing gradually reduces as the flows converge to the detailed balance condition.

The DEBT control protocol is derived in the following steps:

1. We begin by assuming that each pair of nodes that wish to transact has some desired flow rate, but is willing to accept a smaller flow rate as well. In other words, the transaction demands are elastic. To indicate a preference for a larger flow rate, we assign each transacting node-pair with a certain utility that is an increasing function of the flow rate obtained.

2. The goal of the protocol is to solve the following constrained optimization problem: maximize the sum of all the node-pair's utilities, subject to the constraint that flows satisfy the detailed balance condition and the flow rates served do not exceed the desired flow rate.

3. The constraints are relaxed by introducing Lagrange multipliers. An equivalent dual problem and a saddle point problem are derived.

4. An iterative algorithm is proposed to solve the dual problem. The algorithm updates both flow variables (primal variables) and the Lagrange multipliers (dual variables).

5. The above algorithm is shown to have a decentralized implementation. The Lagrange multipliers are interpreted as prices quoted by channels. Based on these prices, transacting

node-pairs make routing and flow-control decisions in a rational fashion.

6. The optimality of the protocol is argued based on the conditions at stationary points of the protocol and the equivalence between the primal, dual and saddle-point formulations.

We elaborate on each of these steps next.

4.4.1 An Optimization Problem

To precisely formulate the protocol's objective, we make the following assumptions regarding the transaction demands made to the PCN. Firstly, we assume that each pair of nodes demands a constant flow of transactions. Put mathematically, for each node pair (i, j), we assume that user i wants to pay user j at a rate of $a_{i,j}$ per unit time, with $a_{i,j} \in \mathbb{R}_+$. If $a_{i,j} = 0$ for some node-pair (i, j), it means that i does not wish to send any money to j. We do not make any assumptions on the values of $\{a_{i,j} : (i, j) \in V \times V\}$. Secondly, we assume that transacting at a slower rate than what they requested is acceptable to the nodes, although a faster rate is preferable. We model this preference by means of a utility function. We assume that a pair of nodes (i, j) gains a certain utility $U_{i,j}(f_{i,j})$ when being served a flow of rate $f_{i,j} \in [0, a_{i,j}]$. We assume that the utility function $U_{i,j}(\cdot)$ concave, continuous and non-decreasing. We also assume that $U_{i,j}(0) = 0$ for all (i, j). The utility functions may be different for different node-pairs. Often, we shall consider one of two special cases:

- linear utility functions, with $U_{i,j}(f_{i,j}) = \nu f_{i,j}$, with ν being some positive constant.
- strongly concave, twice continuously differentiable utility functions with a curvature bounded away from zero on the interval $[0, a_{i,j}]$. Mathematically, this means $U''_{i,j}(\cdot) \leq -\alpha < 0$.

The DEBT control protocol's objective is to determine a set of flows f in steady state that maximizes the sum of utilities of all pairs of transacting nodes, subject to the transaction demand patterns and the detailed balance flow requirement. To elaborate, the goal of the protocol is to maximize $\sum_{(i,j)\in\mathcal{N}} U_{i,j}(f_{i,j})$, subject to the following constraints.

- The total flow from i to j, $f_{i,j}$, is less than or equal to $a_{i,j}$. We call this the demand constraints.
- The net flow along any channel in the network is zero, i.e., $(Rf)_{u,v} = 0$ for all channels (u, v). We call this the detailed balance constraint.

More succinctly, the problem of our interest can be written as:

$$\begin{aligned} \max_{f\geq 0} \quad & \sum_{(i,j)\in\mathcal{N}} U_{i,j}(f_{i,j}) \\ \text{s.t.} \quad & Rf = 0 \\ & f_{i,j} \leq a_{i,j} \quad \forall\ (i,j) \in V \times V \end{aligned} \tag{\textbf{P}}$$

The equation $Rf = 0$ represents the detailed balance constraint and the inequalities $f_{i,j} \leq a_{i,j} \quad \forall\ (i,j)$ represents the demand constraints. For brevity, we denote the set of non-negative flows satisfying the demand constraints by the symbol A.

$$A \triangleq \{f : f \geq 0,\ f_{i,j} \leq a_{i,j} \quad \forall\ (i,j) \in V \times V\} \tag{4.1}$$

The symbol $(\mathbf{P})$ denotes that the optimization problem presented above is the *primal* (or original) problem.

4.4.2 The Dual Problem

We now present an alternate, equivalent form of the primal problem that do not explicitly have the detailed balance constraint. The technique of relaxing constraints using *Lagrange multipliers*, which we now present, is a standard technique in the field of optimization (see the book, Nonlinear Programming, by Bertsekas [55]). Let $\lambda_{u,v}$ denote the Lagrange multiplier for the constraint $(Rf)_{u,v} = 0$; let $\lambda \in \mathbb{R}^E$ denote the vector of all such terms. Define the Lagrangian of the problem $(\mathbf{P})$ by

$$L(f,\lambda) \triangleq \sum_{(i,j)\in\mathcal{N}} U_{i,j}\left(f_{i,j}\right) - \lambda^T Rf. \tag{4.2}$$

Using the Lagrangian function, we can formulate an equivalent form of the problem $(\mathbf{P})$ as follows:

$$\begin{aligned} &\max_{f\in A}\ \min_{\lambda\in\mathbb{R}^E} \quad L(f,\lambda) \\ =&\max_{f\in A}\ \min_{\lambda\in\mathbb{R}^E} \quad \sum_{(i,j)\in\mathcal{N}} U_{i,j}\left(f_{i,j}\right) - \lambda^T Rf \end{aligned} \tag{4.3}$$

Let us denote the inner optimization problem by $(\mathbf{P}^*(f))$:

$$\min_{\lambda\in\mathbb{R}^E} \quad \sum_{(i,j)\in\mathcal{N}} U_{i,j}\left(f_{i,j}\right) - \lambda^T Rf. \tag{$\mathbf{P}^*(f)$}$$

To see why (4.3) is equivalent to $(\mathbf{P})$, consider the problem $(\mathbf{P}^*(f))$. Unless $Rf = 0$, the value of $(\mathbf{P}^*(f))$ is $-\infty$. Therefore, to solve (4.3), f must be chosen such that $Rf = 0$. However, under this constraint, the value of $(\mathbf{P}^*(f))$ is equal to the objective function in $(\mathbf{P})$.

The dual of the optimization problem $(\mathbf{P})$ (or equivalently, of (4.3)) is obtained by changing the

order of the minimization and maximization in (4.3):

$$\begin{aligned} &\min_{\lambda \in \mathbb{R}^E} \max_{f \in A} \quad L(f, \lambda) \\ &= \min_{\lambda \in \mathbb{R}^E} \max_{f \in A} \quad \sum_{(i,j) \in \mathcal{N}} U_{i,j}\left(f_{i,j}\right) - \lambda^T R f \\ &= \min_{\lambda \in \mathbb{R}^E} D(\lambda), \end{aligned} \tag{D}$$

where $D(\lambda)$ is called the dual function, and is defines as follows:

$$D(\lambda) \triangleq \max_{f \in A} L(f, \lambda) \quad \forall\, \lambda \in \mathbb{R}^E. \tag{4.4}$$

Note that (**P**) is a convex optimization problem with linear equality constraints and a bounded, polyhedral domain (composed of linear inequality constraints). Thus, there is a strong relation between the primal problem (**P**) and its dual (**D**) (see [56], Appendix C). In particular, we note the following theorem, which will be useful later in relating the optimal solutions of (**D**) and (**P**).

Theorem 4.1 (Saddle Point Theorem (adapted from [56]))**.** *For f^* to be an optimal solution of* (**P**) *and λ^* to be an optimal solution of* (**D**)*, it is necessary and sufficient that $f^* \in A$ and*

$$L(f^*, \lambda) \geq L(f^*, \lambda^*) \geq L(f, \lambda^*) \qquad \forall\ \lambda \in \mathbb{R}^E,\ \forall\ f \in A. \tag{S}$$

Equation (**S**) is the definition of a saddle-point of $L(f, \lambda)$. Thus, the above statement says that f^* is primal optimal and λ^* is dual optimal if and only if (f^*, λ^*) is a saddle-point for the Lagrangian.

4.4.3 A Dual Algorithm

The dual algorithm we now present is essentially a gradient-descent algorithm to solve the dual problem (**D**). As a first step, we obtain an expression for the gradient of $D(\lambda)$; we do so by invoking Danskin's theorem.

Theorem 4.2 (Danskin's Theorem; from [55], Appendix B and [57])**.** *Suppose $\phi(x, z)$ is a continuous function of two arguments*

$$\phi : \mathbb{R}^n \times Z \to \mathbb{R}$$

where $Z \subset \mathbb{R}^m$ is a convex, compact set. Further, assume $\phi(x, z)$ is convex in x for every $z \in Z$. Define $f(x)$ and $Z_0(x)$ as:

$$f(x) = \max_{z \in Z} \phi(x, z),$$

$$Z_0(x) = arg\ \max_{z \in Z} \phi(x, z).$$

Then the following statements are true:

- *$f(x)$ is convex in x.*

- *If $Z_0(x)$ consists of a single element $\overline{z}$, then $f(x)$ is differentiable at x, with*

$$\nabla f(x) = \nabla_x \phi(x, \overline{z}).$$

- *If $\phi(x, z)$ is differentiable with respect to x for all $z \in Z$, and if $\nabla_x \phi(x, z)$ is continuous with respect to z for all x, then the subdifferential set of $f(x)$ is given by:*

$$\partial f(x) = conv\{\nabla_x \phi(x, z) : z \in Z_0(x)\}$$

where $conv\{\cdot\}$ denotes the convex hull of a set.

Lemma 4.1 (Properties of $D(\lambda)$)**.** *Let $D(\lambda)$ be the dual function of the optimization problem* (**P**) *as defined in* (4.4)*. Then $D(\lambda)$ is a convex function over the entire domain $\mathbb{R}^E$, and is, therefore, continuous on $\mathbb{R}^E$. Moreover, the subdifferential set of $D(\lambda)$ is given by*

$$\partial D(\lambda) = conv\{-Rf : f \in F(\lambda)\},$$

where

$$F(\lambda) \triangleq \arg\max_{f \in A} L(f, \lambda) \tag{4.5}$$

Proof. The claims follow immediately from Danskin's theorem quoted above (Theorem 4.2). Replace x by the Lagrange multipliers λ, z by the flow variables f, and $\phi(x, z)$ by the Lagrangian $L(f, \lambda)$. The set Z corresponds to A, the domain of f; in our setting, this is a compact, convex set.Then $f(x)$ corresponds to the dual function $D(\lambda)$ and $Z_0(x)$ is the set $\arg\max_{f \in A} L(f, \lambda)$. Since $L(f, \lambda)$ is linear in λ for all f, it is also convex; thus, the convexity requirement is satisfied. Lastly, $\nabla_x \phi(x, z)$ corresponds to $\nabla_\lambda L(f, \lambda)$, which is equal to $-Rf$. It immediately follows that $\nabla_\lambda L(f, \lambda)$ is continuous with respect to f. Thus, all conditions for applying Danskin's theorem are met. □

Lemma 4.1 highlights the fact that $D(\lambda)$ is differentiable only at those values of λ where the set $F(\lambda)$ has a unique value. This uniqueness condition may not be met at all points, which means that $D(\lambda)$ may not be differentiable everywhere. However, even in this case, $-Rf(\lambda)$ is a subgradient of $D(\lambda)$, where $f(\lambda)$ is an arbitrary element of $F(\lambda)$.

We now present the algorithm as follows.

$$\begin{aligned} \lambda[0] &= 0 \\ f[t] &\in \arg\max_{f \in A} L(f, \lambda[t]) \\ \lambda[t+1] &= \lambda[t] + \epsilon Rf[t] \end{aligned} \tag{A}$$

Here, ϵ is a strictly positive step-size parameter in the algorithm. The algorithm is an iterative, discrete-time one. The iterations continue for all $t \in \mathbb{N}$. In each iteration, the flows f are set so as to maximize the Lagrangian, given the current values of λ, while the Lagrange multipliers λ take a

step in the direction towards minimizing the Lagrangian L. Under conditions where the set $F(\lambda[t])$ is unique everywhere, algorithm (**A**) is exactly the gradient descent of $D(\lambda)$. Otherwise, it can be interpreted as a subgradient algorithm. In either case, we expect $\lambda[t]$ to approach an optimal solution of (**D**), and consequently, $f[t]$ to approach an optimal solution of (**P**). We analyze the convergence of the algorithm (**A**) in Section 4.5

4.4.4 From a Dual Algorithm to a Network Protocol

We now illustrate how (**A**) can be implemented in a distributed way on a payment channel network. In addition, we show how the Lagrange multiplier $\lambda_{u,v}$ can be interpreted as the price quoted by channel (u, v) to route a unit of flow in the direction $u \to v$. For ease of exposition, we split this discussion into two parts:

- the flow-control and routing decisions by the node-pairs, and
- the price updates by the channels.

To understand how the routing and flow-control decisions are decentralized, consider the optimization problem that describes how flows are set in (**A**):

$$\begin{aligned} f[t] &= \arg\max_{f \in A} L(f, \lambda[t]) \\ &= \arg\max_{f \in A} \sum_{i,j} U_{i,j}(f_{i,j}) - \lambda[t]^T R f \\ &= \arg\max_{f \in A} \sum_{i,j} U_{i,j}(f_{i,j}) - \mu[t]^T f \qquad \left(\mu \triangleq R^T \lambda\right) \\ &= \arg\max_{f \in A} \sum_{i,j} \left(U_{i,j}(f_{i,j}) - \sum_{k=1}^{|P_{i,j}|} \mu_{i,j,k}[t] f_{i,j,k} \right) \end{aligned} \tag{4.6}$$

First, note that in (4.6), the optimization problem is separable into non-interacting components. To elaborate, each pair of transacting nodes (i, j) can solve the following problem independently of the other transacting node-pairs:

$$(f_{i,j,1}[t], \ldots, f_{i,j,|P_{i,j}|}[t]) = \arg\max \; U_{i,j}(f_{i,j}) - \sum_{k=1}^{|P_{i,j}|} \mu_{i,j,k}[t]\, f_{i,j,k}, \quad \text{where} \tag{4.7}$$

$$\mu_{i,j,k}[t] = \sum_{u \to v \in p_{i,j,k}} \lambda_{u,v}[t] - \sum_{v \to u \in p_{i,j,k}} \lambda_{u,v}[t] \tag{4.8}$$

Let $\lambda_{u,v}$ be interpreted as the price per unit flow for routing flows in the direction $u \to v$. Let the price for routing flows in the opposite direction of the same channel be $-\lambda_{u,v}$. Then, $\mu_{i,j,k}$ can be interpreted as the price along the path $p_{i,j,k}$. It is equal to the sum of the prices along each of

the links in the path with the appropriate direction incorporated (see (4.8)). Each of these price terms are evolving with time.

With this interpretation, $f_{i,j,k}\mu_{i,j,k}$ is the total cost of routing an amount $f_{i,j,k}$ along the path $p_{i,j,k}$, so $\sum_{k=1}^{|P_{i,j}|} f_{i,j,k}\mu_{i,j,k}$ is the total cost incurred by the node-pair (i,j) for splitting the total flow $f_{i,j}$ along different paths. It is natural then to see that this cost is subtracted from the total utility gained by the node-pair (i,j). Thus, the interpretation of (4.7) is that each node pair tries to maximize its net utility, i.e., the utility with the cost subtracted. This is a rational decision from the nodes' point of view.

Next, we show how solving (4.7) leads to joint flow-control and routing. The structure of the optimization problem in (4.7) is such that it is optimal to route flows only along the path that has the minimum price. To elaborate, each node pair (i,j) calculates the minimum value and the minimizer of the set $(\mu[t]_{i,j,k} : k = 1, 2, \ldots, |P_{i,j}|)$; denote these quantities as $\mu^*_{i,j}[t]$ and $k^*_{i,j}[t]$ respectively. If there are ties, let $k^*_{i,j}[t]$ be chosen according to some fixed preference order among the paths (say, preferring the shortest path). Then, for each (i,j), the path from i to j indexed by $k^*_{i,j}[t]$ carries the entire flow $f_{i,j}[t]$, and all other paths from i to j carry zero flow; here, $f_{i,j}[t]$ is the solution of

$$\max_{f\in[0,a_{i,j}]} U_{i,j}(f) - f\mu^*_{i,j}[t].$$

This is a one-dimensional optimization problem, whose solution can be explicitly computed as follows. The interpretation of (4.9) is given below.

$$f_{i,j}[t] = \begin{cases} 0 & \text{if } U'_{i,j}(0) \le \mu^*_{i,j}[t] \\ U'^{-1}_{i,j}(\mu^*_{i,j}[t]) & \text{if } U'(0) > \mu^*_{i,j}[t] > U'(a_{i,j}) \\ a_{i,j} & \text{if } \mu^*_{i,j}[t] \le U'(a_{i,j}) \end{cases} \tag{4.9}$$

It is possible that $U_{i,j}(\cdot)$ is not differentiable everywhere. However, since $U_{i,j}(\cdot)$ is a concave function, a subdifferential set exists at each point. For a one-dimensional function such as $U_{i,j}(\cdot)$, the sub-differential is a range of values at points where the function is non-differentiable, and is a single point otherwise. Thus, we can define $U'_{i,j}(\cdot)$ on the entire interval $[0, a_{i,j}]$, with $U'_{i,j}(\cdot)$ taking a range of values at points where $U_{i,j}$ is not differentiable.

Defined this way, $U'_{i,j}(\cdot)$ is guaranteed to be a decreasing function. When $U'_{i,j}(\cdot)$ is a strictly decreasing function, $U'^{-1}_{i,j}(\cdot)$ is well defined. Even if the function is not strictly decreasing (but simply non-increasing), we define $U'^{-1}_{i,j}(\cdot)$ to be a set and set $f^*_{i,j}[t]$ to be the maximum value within that set. Note that when $U_{i,j}(\cdot)$ is linear, the optimal solution is to either route the entire requested amount $a_{i,j}$ or none of it. Thus, linear utility functions, automatically lead to atomicity of the transactions.

Next, we turn our attention to the updates of the Lagrange multipliers in (**A**). The updates for

each variable can be written out explicitly as

$$\lambda_{u,v}[t+1] = \lambda_{u,v}[t] + \epsilon(Rf)_{u,v}[t] \qquad \forall\ (u,v) \in E \tag{4.10}$$

Here, ϵ is a step-size for the Lagrange multiplier increments; it's a parameter of the algorithm. Note that the price $\lambda_{u,v}$ is updated at time step t based only on the net flow through the channel (u,v) in slot t. Thus, the price updates are based on local information alone. In fact, channels need not know the source or the destination of the flows that they are serving to calculate prices. The term $\lambda_{u,v}$ tends to increase if the net flow through the channel (u,v) is in the direction from $u \to v$, and tends to decrease if the net flow is in the opposite direction. This change is consistent with the interpretation of $\lambda_{u,v}$ as a price that penalizes or encourages flows in order to maintain detailed balance. Indeed, a sustained net flow in either direction is bound to increase the price for any future flow in that direction. It also decreases the price for any flow in the opposite direction, thereby encouraging such flows. The prices keep adjusting until the net flow through each channel converges to zero.

We now show how algorithm (**A**) can be overlaid on the model of the payment channel network presented in Section 4.3. Firstly, each iteration of algorithm (**A**) corresponds to a single time step of the payment channel network. $f[t]$ denotes the flow requests made in time step t, while $\lambda[t]$ denotes the prices quoted by the channels at the beginning of time step t. Within a time step t, transacting node-pairs make flow-control and routing decisions based on the prices quoted by the channels at the beginning of the time step; i.e, $f[t]$ is computed from $\lambda[t]$. The prices are updated at the end of a time slot, after the flows have been routed. Thus, $\lambda[t+1]$ is computed after $f[t]$ has been routed.

We have already noted that each node-pair solves its own optimization problem to decide its flow, without interacting with the other node-pairs. Having made their decisions, node-pairs route their flow requests on the network. If the flow vector is feasible, all transaction requests go through, and channels update their balances as well as their prices. If some channels are unable to route the flows that have been requested through them, they rebalance themselves and allow the flows to pass through. Under the assumption that the PCN is in the large capacity regime (see Section 4.3.4), the requested flows will be feasible after this rebalancing step. Note that the price updates does not depend on whether channels rebalance or not. Each channel updates its own price and makes the price publicly available at the end of each slot.

Assuming that the protocol converges to a detailed balance flow, the frequency with which a channel needs to rebalance reduces, eventually approaching zero. If the protocol converges fast enough, the number of times a channel would need to rebalance is finite. In fact, if the channel capacities are large enough, the channels would not need to rebalance at all. Moreover, the protocol never runs into deadlocks. This is because, in the transient phase, the channels rebalance themselves if they get imbalanced, while in steady-state, the protocol stifles out deadlock-causing flows.

4.5 Convergence Analysis

4.5.1 Preliminary Results

As the first step in the convergence analysis, we specify conditions under which $D(\lambda)$ is differentiable everywhere.

- **Assumption 1:** For every pair of nodes (i, j), there is exactly one path along which transactions may be routed from node i to node j.
- **Assumption 2:** For every pair of nodes (i, j), the utility function $U_{i,j}(\cdot)$ is strictly concave and continuously differentiable in the interval $[0, a_{i,j}]$.

Lemma 4.2 (Gradient of $D(\lambda)$)**.** *Under **Assumption 1** and **Assumption 2**, the following statements hold:*

- *For all $\lambda \in \mathbb{R}^E$, the set $F(\lambda)$ has exactly one element $f(\lambda)$ ($F(\lambda)$ defined in (4.5)).*
- *$f(\lambda)$ is a continuous function of λ.*
- *$D(\lambda)$ is continuously differentiable with $\nabla D(\lambda) = -Rf(\lambda)$.*

Proof. Recall the discussion on the flow updates in Section 4.4.4. We already know that the flows for each (i, j) pair are set independently of the others. By **Assumption 1**, only the value of $f_{i,j}$ needs to be decided; the question of splitting the total flow along multiple paths does not arise. **Assumption 2** implies that the function $U'_{i,j}(\cdot)$ is strictly decreasing over the interval $[0, a_{i,j}]$. This, in turn, implies that $U'^{-1}_{i,j}(\cdot)$ is a well-defined (i.e., single-valued) function over all real values. The first claim then follows from (4.9).

Under **Assumption 2**, the functions $U'_{i,j}(\cdot)$ are continuous. Thus, so are the functions $U'^{-1}_{i,j}(\cdot)$. These are the functions that map the path prices to the flows. The path prices are simply the sum of a few channel prices. Thus, the flows are a continuous function of the channel prices, i.e., $f(\lambda)$ is a continuous function of λ. This proves the second statement. For the third statement, the differentiability of $D(\lambda)$ and the expression of $\nabla D(\lambda)$ follows immediately from Lemma 4.1, while the continuity of its derivative follows from the second statement. □

Lemma 4.2 establishes that under **Assumption 1** and **Assumption 2**, the iterates of λ in (**A**) are identical to the iterates of the gradient descent method on $D(\lambda)$. Note that even without these assumptions, the iterates perform sub-gradient descent; however, we shall not analyze this general case here.

4.5.2 Convergence in the Continuous-Time Limit

In this section, we analyze the algorithm under the limit $\epsilon \to 0$, while speeding time by a factor γ/ϵ for some fixed $\gamma > 0$. In this limiting regime, algorithm (**A**) can be described by the following

differential equation system:

$$f(t) = f(\lambda(t)) = \arg \max_{f \in A} L(f, \lambda(t)) \tag{4.11}$$

$$\dot{\lambda}(t) = \gamma R f(t) \; ; \gamma > 0 \tag{4.12}$$

A shorthand representation of the above dynamical system is given by

$$\dot{\lambda} = \gamma R f(\lambda) \tag{4.13}$$

We would like to show that $\lambda(t)$ converges to an optimal solution of (**D**), and consequently, $f(t)$ converges to an optimal value of (**P**). The main tool we shall use to prove such a convergence result is LaSalle's invariance principle.

Theorem 4.3 (La Salle's Invariance Principle; from [58] and [59])**.** *Consider a system with the dynamics $\dot{x} = f(x)$. Let Ω be a compact positively invariant set with respect to the system dynamics. Let V be a real-valued continuously differentiable function such that $\dot{V}(x) \leq 0$ in Ω, where $\dot{V}(x) \triangleq [\nabla V(x)]^T f(x)$. Let E be the set of all points in Ω where $\dot{V}(x) = 0$. Let $M \subseteq E$ be the largest invariant set in E. Then every trajectory starting in Ω approaches M as $t \to \infty$, i.e.,*

$$\lim_{t \to \infty} \left\{ \inf_{z \in M} \|x(t) - z\| \right\} = 0.$$

For the problem at hand, $D(\lambda)$ serves as a potential function, i.e., a function whose value is decreasing (or non-increasing) with the dynamics of λ as given in (4.13). Indeed, we have:

$$\begin{aligned} \dot{D}(\lambda) &= \nabla D(\lambda)^T \dot{\lambda} \\ &= -(Rf(\lambda))^T Rf(\lambda) \\ &= -\|Rf(\lambda)\|^2 \leq 0 \end{aligned}$$

The main challenge in applying La Salle's invariance principle to the problem at hand is to find a compact positively invariant set with respect to the dynamics (4.13). To circumvent this issue, we work with a regularized version of the dual function. Let $c > 0$ be some large positive constant; we will specify how large it should be shortly. Define

$$D_c(\lambda) \triangleq D(\lambda) + \sum_{(u,v) \in E} (|\lambda_{u,v}| - c)_+^2, \tag{4.14}$$

where the notation x_+ denotes $\max(x, 0)$. In addition, we modify the system dynamics such that λ decreases along the gradient of $D_c(\lambda)$, instead of following the gradient of $D(\lambda)$. The new system dynamics are given by

$$\dot{\lambda} = -\nabla D_c(\lambda) = Rf(\lambda) - \text{sgn}(\lambda)(|\lambda| - c)_+, \tag{4.15}$$

where $\text{sgn}(\lambda)(|\lambda| - c)_+$ denotes a vector indexed by the set of channels, E, whose (u, v) component is $\text{sgn}(\lambda_{u,v})(|\lambda_{u,v}| - c)_+$. Note that the modified dynamics are identical to the original one over the set $\{\lambda : \|\lambda\|_\infty \leq c\}$. Only when some components of λ grow very large in magnitude, the regularizer term kicks in, keeping them bounded.

Lemma 4.3. *Let $c > 0$ be fixed. For any $\alpha > 0$, the set $\{\lambda : D_c(\lambda) \leq \alpha\}$ is a compact positively invariant set under the evolution of λ under* (4.15)*. Moreover, starting from $\lambda(0) = 0$ and evolving according to* (4.15)*, the trajectory of $\lambda(t)$ converges to the set M_c as $t \to \infty$, where $M_c \triangleq \{\lambda : \nabla D_c(\lambda) = 0\}$.*

Proof. For any $\alpha > 0$, call the set $\{\lambda : D_c(\lambda) \leq \alpha\}$ a *level set* of $D_c(\lambda)$. Since $D_c(\lambda)$ is a continuous function of λ, its level sets are closed sets. Thus, to show that the level sets are compact, it suffices to show that it is bounded. Firstly, note that $D(\lambda) \geq 0$. This is because $D(\lambda)$ is the maximum value of $L(f, \lambda)$ over $f \in A$ (see (4.4)) and $L(0, \lambda) = 0$ for all λ. Secondly, the level sets of the regularizer alone, $\sum_{(u,v)\in E}(|\lambda_{u,v}| - c)_+^2$, are bounded. Adding a non-negative term, $D(\lambda)$, would only shrink the level sets. Thus, the level sets of $D_c(\lambda)$ are bounded.

The positive invariance property of the level sets follows from the fact that $D_c(\lambda)$ monotonically decreases with the system dynamics. This is shown below:

$$\begin{aligned}
\dot{D}_c(\lambda) &= \nabla D_c(\lambda)^T \dot{\lambda} \qquad \text{(by the chain rule)} \\
&= -\|\nabla D_c(\lambda)\|^2 \qquad \text{(by (4.15))} \\
&\leq 0
\end{aligned}$$

We now have all the ingredients to invoke LaSalle's invariance theorem (Theorem 4.3). The function $D_c(\lambda)$ serves as the Lyapunov (or potential) function. Let $\alpha \triangleq D_c(0)$ (which is also equal to $D(0)$). Then $\Omega_\alpha \triangleq \{\lambda : D_c(\lambda) \leq \alpha\}$ is a compact, positively invariant set. We know that $\lambda(0) \in \Omega_\alpha$, which implies $\lambda(t) \in \Omega_\alpha \; \forall \; t \to \infty$. We also know that the set $\{\lambda : \dot{D}_c(\lambda) = 0\}$ is equal to M_c. Moreover, every $\lambda \in M_c$ is also a stationary point of the dynamics. Thus, by LaSalle's invariance principle,

$$\lambda(t) \xrightarrow{t\to\infty} M_c \cap \Omega_\alpha \quad \Rightarrow \lambda(t) \xrightarrow{t\to\infty} M_c$$

□

So far, we have shown that, under the regularized dynamics of (4.15), λ converges to M_c, the set of minimizers of the regularized dual function $D_c(\lambda)$. Let us summarize the assumptions and approximations that we have made in proving this result:

- We assume that each pair of nodes route transactions on a single path and that they have strictly concave, twice differentiable utility functions (see Section 4.5.1)
- We approximate the updates of the variables under algorithm (**A**) by the corresponding continuous-time dynamics (4.13).

- We modify the dynamics to ensure that λ does not grow too large (4.15).

By making an additional assumption, namely, a (finite) minimizer of $D(\lambda)$ exists, we can obtain more meaningful convergence guarantees. We elaborate on this below.

Assumption 3: The function $D(\lambda)$ achieves its minimum for some (finite) value of λ.

Define Λ^* to be the set of minimizers of $D(\lambda)$, i.e.,

$$\Lambda^* \triangleq \arg\min_{\lambda \in \mathbb{R}^E} D(\lambda).$$

Further, define c^* to be the least possible norm of a minimizing λ, i.e.,

$$c^* \triangleq \inf\{\|\lambda\|_\infty : \lambda \in \Lambda^*\}.$$

Clearly, under **Assumption 3**, Λ^* is non-empty and c^* is finite.

The following lemma refines the statement of Lemma 4.3 with the additional condition that **Assumption 3** is true.

Lemma 4.4. *Suppose **Assumption 3** is true. Let c be chosen s.t. $c > c^*$. Then, $M_c \subseteq \Lambda^*$. Further, $\lambda(t) \to \Lambda^*$ under the dynamics of* (4.15).

Proof. The function $D(\lambda)$ is a convex function of λ (see Lemma 4.1). The regularization term, $\sum_{u,v}(|\lambda_{u,v}| - c|)_+^2$, is also a convex function of λ. Thus, $D_c(\lambda)$ is a convex function. We know that for convex functions defined over the whole space, a particular point is a minimizer of the function if and only if the gradient at that point is zero. Recall that the set M_c consists of all λ such that $\nabla D_c(\lambda) = 0$. Therefore, M_c is the set of minimizers of $D_c(\lambda)$.

Under **Assumption 3**, we can choose some $\lambda^* \in \Lambda^*$ with $\|\lambda^*\|_\infty \leq c$ (since $c > c^*$). Therefore, the dual function is equal to its regularized counterpart at λ^*, i.e., $D(\lambda^*) = D_c(\lambda^*)$. Combining this with the fact that λ^* is a minimizer of $D(\lambda)$, we see that λ^* is also a minimizer of $D_c(\lambda)$. Indeed, any other choice of λ cannot reduce the value of either $D(\lambda)$ or the regularizer term any further.

Extending the same argument, we can make the stronger claim that, under **Assumption 3** and for every $c > c^*$, every minimizer of $D_c(\lambda)$ is also a minimizer of $D(\lambda)$. Thus, $M_c \subseteq \Lambda^*$. By Lemma 4.3, $\lambda(t) \to M_c$. Therefore, $\lambda(t) \to \Lambda^*$. □

Lemma 4.4 tells us that, under appropriate conditions, the prices, $lambda(t)$, converge to the set of optimal solutions of the dual problem (**D**). Building upon this result, we can show that the flows, $f(t)$, converge to the set of optimal solutions of the original network optimization problem (**P**). Let F^* denote the set of solutions to (**P**).

Lemma 4.5. *Suppose **Assumption 3** is true and suppose c is chosen such that $c > c^*$. Then, under the regularized dynamics,* (4.15), *$f(\lambda(t)) \to F^*$.*

Proof. Let λ^* be any optimal solution of (**D**). Let f^* denote $f(\lambda^*)$, i.e., the flow that maximizes the Lagrangian given, given the prices are λ^*. Since $D(\lambda)$ is a convex function, it must be the case that $\nabla D(\lambda^*) = Rf^* = 0$. This implies that $L(f^*, \lambda)$ does not depend on λ at all. In other words,

$$L(f^*, \lambda) = \sum_{i,j} U_{i,j}(f^*_{i,j}) - \lambda^T R f^* = L(f^*, \lambda^*) \ \forall \ \lambda \in \mathbb{R}^E$$

In addition, since f^* is the best flow in response to λ, i.e., $f^* = \arg\max_{f \in A} L(f, \lambda)$, we get:

$$L(f^*, \lambda^*) \geq L(f, \lambda^*) \ \forall f \in A$$

Putting these equations together, we have

$$L(f^*, \lambda) = L(f^*, \lambda^*) \geq L(f, \lambda^*) \ \forall \ \lambda \in \mathbb{R}^E, \ \forall f \in A$$

Thus, (f^*, λ^*) form a saddle point. By the saddle point theorem (Theorem 4.1), f^* must be an optimal solution of the primal problem (**P**).

Under **Assumption 3** and the condition $c > c^*$, Lemma 4.5 tells us that $\lambda(t) \to \Lambda^*$. The above analysis shows that $\{f(\lambda) : \lambda \in \Lambda^*\} \subseteq F^*$. Since $f(\lambda)$ is a continuous function of λ (shown in Lemma 4.2), we obtain the stated convergence result, i.e., $f(\lambda(t)) \to F^*$. □

A key limitation of our convergence results, namely Lemmas 4.4 and 4.5, is that we need to explicitly assume the existence of a solution to the dual problem (**D**) (**Assumption 3**). We conjecture that this assumption is true for all instances of the primal problem (**P**) and the corresponding (**D**). Proving (or disproving) this conjecture is left for future work. Proving similar convergence results without the regularized dynamics would be an additional improvement of our current results.

4.5.3 Convergence with non-zero step-sizes

To establish convergence of the algorithm under step-sizes bounded away from zero, we need to have that $D(\lambda)$ has Lipschitz gradients. This condition is obtained when strengthen **Assumption 2** to require that the functions $U_{i,j}(\cdot)$ are strongly concave. In other words, we require that the curvature of the utility functions is bounded away from zero: $U''_{i,j}(\cdot) \leq -\alpha < 0$. Here, we sketch the analysis by Low and Lapsey [60], which studies a network optimization algorithm that is very similar to the one posed here. The only difference is that in their work, the constraints on the flows are capacity constraints instead of detailed balance constraints, i.e., they are inequality constraints instead of equality constraints. The protocol proposed and analyzed by Low and Lapsey [60] is otherwise identical to the one proposed here. In particular, they obtain a Lipschitz constant for the gradients of D by bounding the matrix norm of its Hessian. The expression for the Hessian and the bounds that follow are identical to our setting. From Lemmas 2 and 3 in their paper, it follows that $D(\lambda)$ has Lipschitz gradients.

It is a well-known result that if a differentiable, convex function has Lipschitz gradients, then for small enough step-sizes, the gradient-descent method converges in the sense $\lim_{t\to\infty} \nabla D(\lambda[t]) = -Rf[t] = 0$ (see the book Parallel and Distributed Computing by Bertsekas and Tsitsiklis [56], Chapter 3, Proposition 2.1). In general, the above convergence result does not imply that the sequence $(f[t], \lambda[t])$ converges. In particular, unlike in [60], the level set $\{\lambda : D(\lambda) \leq D(\lambda[0])\}$ is unbounded. However, if $(f[t], \lambda[t])$ does converge, its converges to (f^*, λ^*) that satisfies (**S**), and is, therefore, primal-dual optimal.

4.6 Simulation Results

4.6.1 Dynamic Routing Example

First, we run algorithm (**A**) on the PCN described in Section 4.2.4. Due to the symmetric nature of the set-up, we only describe the flows from $A \to B$ and the corresponding path prices. Recall that there are two possible paths to route such flows: a short path directly along the channel $A - B$, and a longer path via C. According to algorithm (**A**), nodes choose to route flows along the cheaper path at all times. If the prices are identical, they choose the shorter of the two paths. We see in Figure 4.1 that the protocol follows a periodic pattern, choosing the shorter path twice in succession followed by choosing the longer path once. The prices vary accordingly. The step-size for the prices is set to be $\epsilon = 0.1$. Linear utility functions are used. The choice of the utility function is somewhat immaterial in this example, because the need for flow-control does not arise.

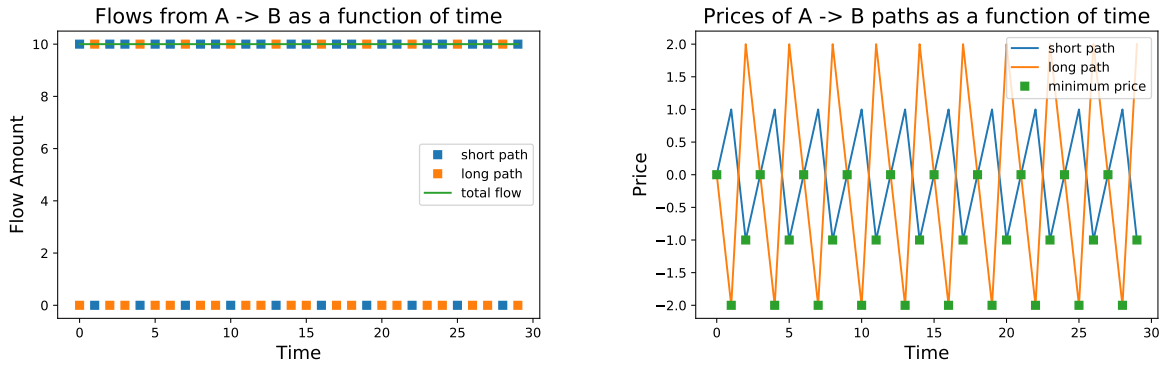

Figure 4.1

Note that although the long-term average of the flows satisfy the detailed balance on each of the edges (accounting for the $B \to C$ and $C \to A$ flows as well), the same cannot be said for the instantaneous flows. In order to get the flows (and the prices) to converge, one can adopt the following heuristic modification of (**A**). Instead of routing the entire flow amount on the path with the minimum cost, one could weigh the paths according to the softmin weights of the path prices,

and split the flows on the various paths accordingly. This smoothens the vector of optimal flows, $f(\lambda)$, as a function of the price vector λ. This in turn helps smoothen $D(\lambda)$, which is a sufficient condition for the gradient descent method with constant step-sizes to converge. We illustrate this point in Figure 4.2. The protocol is run with a step-size of $\epsilon = 0.01$. Softmin is taken using the transformation $x \to \exp(-10x)$.

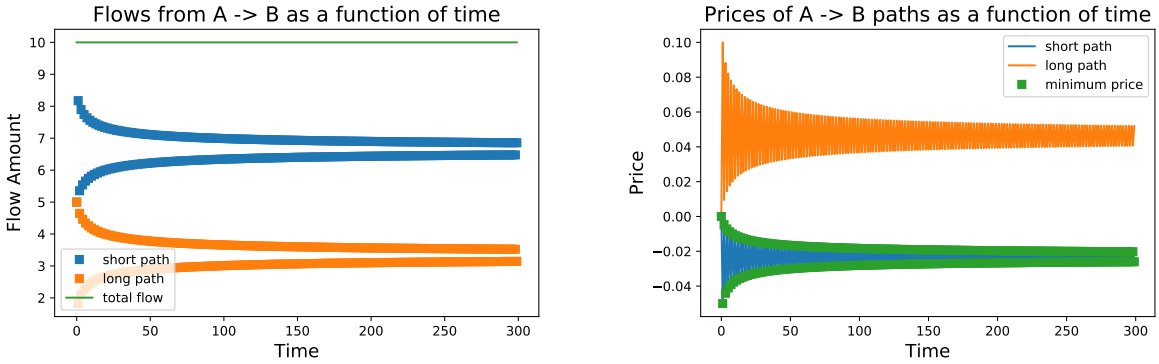

Figure 4.2

4.6.2 Deadlock Example

We now examine the behavior of algorithm (**A**) on the deadlock example presented in Section 4.2.5. In this example, there is no question of routing. The only control exercised by the protocol is that of flow-control. Consequently, the utility function that we choose becomes important. Here, the utility functions are linear with a coefficient of 3. This means that for any path, if the price is strictly above 3, the flow on the path drop to zero. The step-size for the prices is set to be $\epsilon = 0.1$. The simulation results are given in Figure 4.3. We see that with every time step, the path price for the $B->A$ and $B->C$ keeps increasing, until at time-step four, the price exceeds the threshold. At this point, the corresponding flows drop to zero, which, in turn, keeps the prices stable. The prices for the flows from $A \to C$ and $C \to A$ remain zero throughout. Consider the path $A \to B \to C$. The channel price for $A - B$ in the $A \to B$ direction is negative, and for $B - C$ in the $B \to C$ direction is positive. Due to the symmetry of the set-up, the prices add up to zero at all times. The same holds true for flows in the opposite direction. Thus, $A \to C$ and $C \to A$ flows continue throughout, as desired.

4.6.3 Six-Node Network Example

We now consider a payment channel network with six nodes and six payment channels. The topology is that of a ring, as shown in Figure 4.4. The channel capacities are roughly two hundred each. There are demands between many pairs of nodes: these demands are also illustrated in Figure

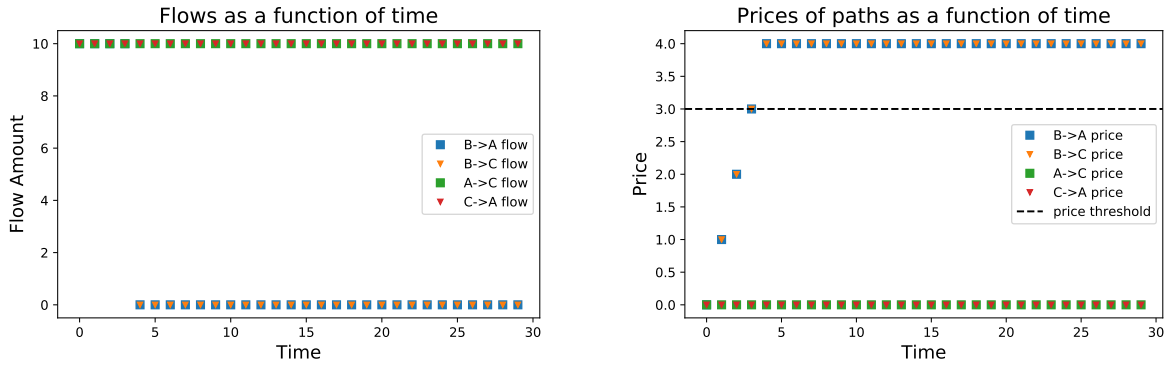

Figure 4.3: A figure with two minipages

4.4.

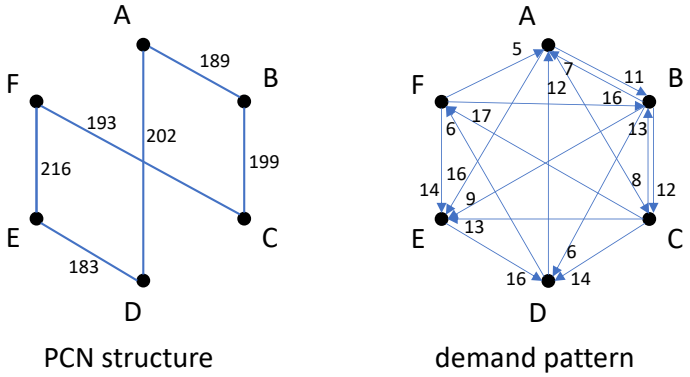

Figure 4.4: Six-Node Network Example

In the simulation, we consider two possible paths for every transacting node-pair: the shortest path, and the shortest path that goes through a randomly chosen node (we allow for cycles here). The choice between paths is made using a softmin function, which is implemented using the transformation $x \to \exp(-x)$. The utility functions are described implicitly by specifying the response of the flows to the path prices (see (4.9)). The response is characterized by two threshold parameters τ_0 and τ_1 with $\tau_0 < \tau_1$.

$$f_{i,j}[t] = \begin{cases} 0 & \text{if } \mu^*_{i,j}[t] \geq \tau_1 \\ a_{i,j}\frac{\tau_1 - \mu^*_{i,j}[t]}{\tau_1 - \tau_0} & \text{if } \tau_1 > \mu^*_{i,j}[t] > \tau_1 \\ a_{i,j} & \text{if } \mu^*_{i,j}[t] \leq \tau_0 \end{cases}$$

The softmin choice of paths and a smooth, strongly concave utility function together provide conditions for the protocol to converge for small enough step-sizes (see Figure 4.6). However, as we

see by comparing Figures 4.5 and 4.6, it may be more desirable to choose a slightly bigger step-size so as to approach the optimal point faster; oscillations around the optimal operating point do not matter.

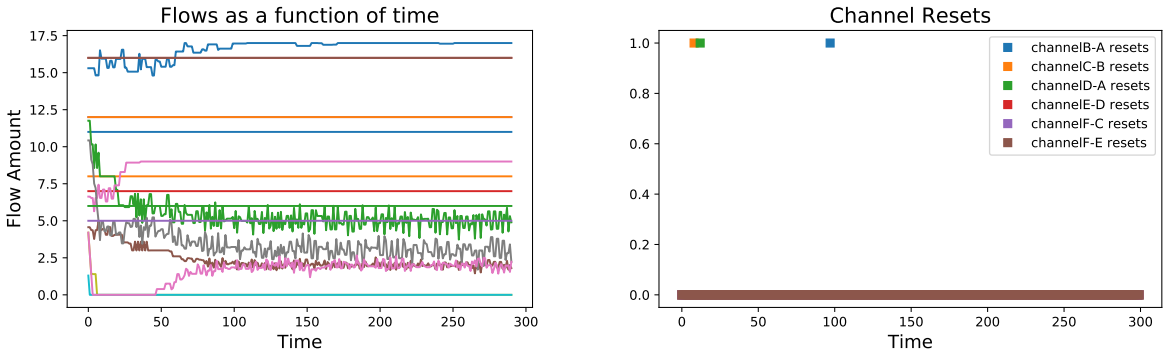

Figure 4.5: $\epsilon = 0.05$. The flows are shown with a moving-average filter with a window of size 10. Note that all the flows do not converge; some of them oscillate around a particular value. The figure on the right illustrates that three channels require rebalancing, once each.

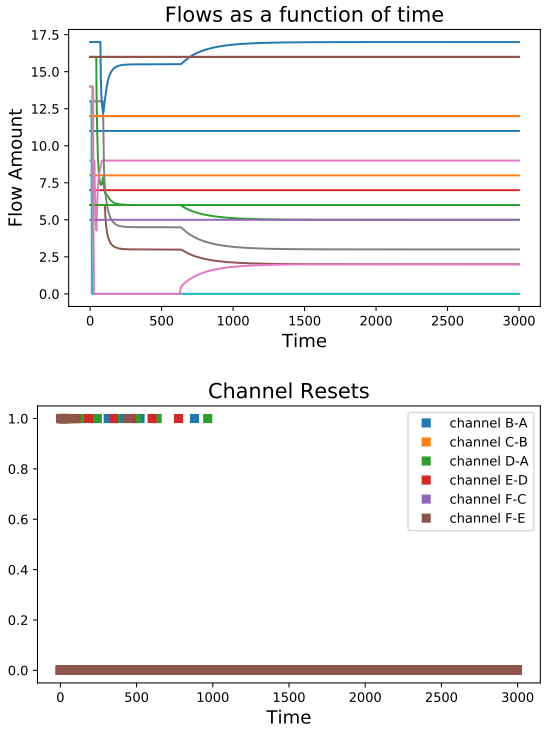

Figure 4.6: $\epsilon = 0.005$. Here, all the flows do converge to a steady-state value, but slowly. As a result, the transient phase lasts much longer, and the channels need to go undergo on-chain rebalancing multiple times.

References

[1] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008.

[2] A. Narayanan, J. Bonneau, E. Felten, A. Miller, and S. Goldfeder, *Bitcoin and cryptocurrency technologies: a comprehensive introduction*. Princeton University Press, 2016.

[3] J. Katz and Y. Lindell, *Introduction to modern cryptography*. CRC press, 2020.

[4] J. Garay, A. Kiayias, and N. Leonardos, "The bitcoin backbone protocol: Analysis and applications," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2015, pp. 281–310.

[5] Y. Sompolinsky and A. Zohar, "Secure high-rate transaction processing in bitcoin," in *International Conference on Financial Cryptography and Data Security*. Springer, 2015, pp. 507–527.

[6] R. Pass, L. Seeman, and A. Shelat, "Analysis of the blockchain protocol in asynchronous networks," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2017, pp. 643–673.

[7] A. Dembo, S. Kannan, E. N. Tas, D. Tse, P. Viswanath, X. Wang, and O. Zeitouni, "Everything is a race and nakamoto always wins," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, p. 859–878.

[8] P. Gaži, A. Kiayias, and A. Russell, "Tight consistency bounds for bitcoin," in *Proceedings of the 2020 ACM SIGSAC Conference on Computer and Communications Security*, 2020, pp. 819–838.

[9] J. Li, D. Guo, and L. Ren, "Close latency–security trade-off for the nakamoto consensus," in *Proceedings of the 3rd ACM Conference on Advances in Financial Technologies*, 2021.

[10] P. Gaži, L. Ren, and A. Russell, "Practical settlement bounds for proof-of-work blockchains," *Cryptology ePrint Archive*, 2021.

[11] L. Gudgeon, P. Moreno-Sanchez, S. Roos, P. McCorry, and A. Gervais, "Sok: Layer-two blockchain protocols," in *International Conference on Financial Cryptography and Data Security*. Springer, 2020, pp. 201–226.

[12] M. Castro and B. Liskov, "Practical byzantine fault tolerance," in *Proceedings of the Third Symposium on Operating Systems Design and Implementation*, ser. OSDI '99. USA: USENIX Association, 1999, p. 173–186.

[13] A. Lewis-Pye and T. Roughgarden, "Resource pools and the cap theorem," *arXiv preprint arXiv:2006.10698*, 2020.

[14] V. Bagaria, S. Kannan, D. Tse, G. Fanti, and P. Viswanath, "Prism: Deconstructing the blockchain to approach physical limits," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 585–602.

[15] J. Wilser, "The lightning network is going to change how you think about bitcoin," https://www.coindesk.com/lightning-network-how-you-think-about-bitcoin, 2021.

[16] S. M. Varma and S. T. Maguluri, "Throughput optimal routing in blockchain based payment systems," *IEEE Transactions on Control of Network Systems*, 2021.

[17] V. Sivaraman, W. Tang, S. B. Venkatakrishnan, G. Fanti, and M. Alizadeh, "The effect of network topology on credit network throughput," *arXiv preprint arXiv:2103.03288*, 2021.

[18] S. Bano, A. Sonnino, M. Al-Bassam, S. Azouvi, P. McCorry, S. Meiklejohn, and G. Danezis, "SoK: Consensus in the age of blockchains," in *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, 2019, pp. 183–198.

[19] J. Garay and A. Kiayias, "SoK: A consensus taxonomy in the blockchain era," in *Cryptographers' Track at the RSA Conference*. Springer, 2020, pp. 284–318.

[20] A. Kiayias, A. Russell, B. David, and R. Oliynykov, "Ouroboros: A provably secure proof-of-stake blockchain protocol," in *Annual International Cryptology Conference*. Springer, 2017, pp. 357–388.

[21] B. David, P. Gaži, A. Kiayias, and A. Russell, "Ouroboros praos: An adaptively-secure, semi-synchronous proof-of-stake blockchain," in *Annual International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2018, pp. 66–98.

[22] C. Badertscher, P. Gaži, A. Kiayias, A. Russell, and V. Zikas, "Ouroboros genesis: Composable proof-of-stake blockchains with dynamic availability," in *Proceedings of the 2018 ACM SIGSAC Conference on Computer and Communications Security*. ACM, 2018, pp. 913–930.

[23] R. Pass and E. Shi, "The sleepy model of consensus," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2017, pp. 380–409.

[24] C. Decker and R. Wattenhofer, "Information propagation in the bitcoin network," in *IEEE P2P 2013 Proceedings*. IEEE, 2013, pp. 1–10.

[25] E. Blum, A. Kiayias, C. Moore, S. Quader, and A. Russell, "The combinatorics of the longest-chain rule: Linear consistency for proof-of-stake blockchains," in *Proceedings of the Fourteenth Annual ACM-SIAM Symposium on Discrete Algorithms*. SIAM, 2020. [Online]. Available: https://epubs.siam.org/doi/pdf/10.1137/1.9781611975994.69 pp. 1135–1154.

[26] C. Dwork, N. Lynch, and L. Stockmeyer, "Consensus in the presence of partial synchrony," *Journal of the ACM (JACM)*, vol. 35, no. 2, pp. 288–323, 1988.

[27] J. Neu, E. N. Tas, and D. Tse, "Ebb-and-flow protocols: A resolution of the availability-finality dilemma," *arXiv preprint arXiv:2009.04987*, 2020.

[28] G. Fanti, J. Jiao, A. Makkuva, S. Oh, R. Rana, and P. Viswanath, "Barracuda: The power of l-polling in proof-of-stake blockchains," in *Proceedings of the Twentieth ACM International Symposium on Mobile Ad Hoc Networking and Computing*, 2019, pp. 351–360.

[29] A. Gopalan, A. Sankararaman, A. Walid, and S. Vishwanath, "Stability and scalability of blockchain systems," *Proc. ACM Meas. Anal. Comput. Syst.*, vol. 4, no. 2, 2020.

[30] L. Ren, "Analysis of nakamoto consensus." *IACR Cryptol. ePrint Arch.*, 2019.

[31] J. Kingman, "A martingale inequality in the theory of queues," *Cambridge Phiolos. Soc.*, vol. 59, pp. 359–361, 1964.

[32] W. Hoeffding, "Probability inequalities for sums of bounded random variables," *Journal of the American Statistical Association*, vol. 58, pp. 13–30, 1963.

[33] M. Yin, D. Malkhi, M. K. Reiter, G. G. Gueta, and I. Abraham, "Hotstuff: BFT consensus with linearity and responsiveness," in *Proceedings of the 2019 ACM Symposium on Principles of Distributed Computing*, 2019, pp. 347–356.

[34] S. Gilbert and N. Lynch, "Brewer's conjecture and the feasibility of consistent, available, partition-tolerant web services," *Acm Sigact News*, vol. 33, no. 2, pp. 51–59, 2002.

[35] S. Gilbert and N. Lynch, "Perspectives on the cap theorem," *Computer*, vol. 45, no. 2, pp. 30–36, 2012.

[36] T. Dinsdale-Young, B. Magri, C. Matt, J. B. Nielsen, and D. Tschudi, "Afgjort: A partially synchronous finality layer for blockchains," in *Security and Cryptography for Networks (SCN)*, 2020.

[37] A. Stewart and E. Kokoris-Kogias, "Grandpa: a byzantine finality gadget," *arXiv preprint arXiv:2007.01560*, 2020.

[38] D. Karakostas and A. Kiayias, "Securing proof-of-work ledgers via checkpointing," Cryptology ePrint Archive, Report 2020/173, 2020, https://eprint.iacr.org/2020/173.

[39] M. Fitzi, P. Gazi, A. Kiayias, and A. Russell, "Parallel chains: Improving throughput and latency of blockchain protocols via parallel composition." *IACR Cryptol. ePrint Arch.*, vol. 2018, p. 1119, 2018.

[40] J. Chen, S. Gorbunov, S. Micali, and G. Vlachos, "Algorand agreement: Super fast and partition resilient byzantine agreement." *IACR Cryptol. ePrint Arch.*, 2018.

[41] P. Daian, R. Pass, and E. Shi, "Snow white: Robustly reconfigurable consensus and applications to provably secure proof of stake," in *International Conference on Financial Cryptography and Data Security*. Springer, 2019, pp. 23–41.

[42] J. Chen and S. Micali, "Algorand," *arXiv preprint arXiv:1607.01341*, 2016.

[43] D. Malkhi, K. Nayak, and L. Ren, "Flexible byzantine fault tolerance," in *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, 2019, pp. 1041–1053.

[44] R. Pass and E. Shi, "Hybrid consensus: Efficient consensus in the permissionless model," in *31st International Symposium on Distributed Computing (DISC 2017)*. Schloss Dagstuhl-Leibniz-Zentrum fuer Informatik, 2017.

[45] V. Buterin and V. Griffith, "Casper the friendly finality gadget," *arXiv preprint arXiv:1710.09437*, 2017.

[46] V. Buterin, D. Hernandez, T. Kamphefner, K. Pham, Z. Qiao, D. Ryan, J. Sin, Y. Wang, and Y. X. Zhang, "Combining ghost and casper," *arXiv preprint arXiv:2003.03052*, 2020.

[47] B. Y. Chan and E. Shi, "Streamlet: Textbook streamlined blockchains." *IACR Cryptol. ePrint Arch.*, vol. 2020, p. 88, 2020.

[48] R. Srikant and L. Ying, *Communication networks: an optimization, control, and stochastic networks perspective*. Cambridge University Press, 2013.

[49] F. Kelly and E. Yudovina, *Stochastic networks*. Cambridge University Press, 2014.

[50] S. Tikhomirov, "Security and privacy of blockchain protocols and applications," Ph.D. dissertation, University of Luxembourg, Esch-sur-Alzette, Luxembourg, 2020.

[51] C. Decker and R. Wattenhofer, "A fast and scalable payment network with bitcoin duplex micropayment channels," in *Symposium on Self-Stabilizing Systems*. Springer, 2015, pp. 3–18.

[52] J. Poon and T. Dryja, "The bitcoin lightning network: Scalable off-chain instant payments," 2016.

[53] V. Bagaria, J. Neu, and D. Tse, "Boomerang: Redundancy improves latency and throughput in payment-channel networks," in *International Conference on Financial Cryptography and Data Security*. Springer, 2020, pp. 304–324.

[54] V. Sivaraman, S. B. Venkatakrishnan, K. Ruan, P. Negi, L. Yang, R. Mittal, G. Fanti, and M. Alizadeh, "High throughput cryptocurrency routing in payment channel networks," in *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI)*, 2020, pp. 777–796.

[55] D. Bertsekas and J. Tsitsiklis, *Nonlinear Programming*. Athena Scientific, 1999.

[56] D. Bertsekas and J. Tsitsiklis, *Parallel and distributed computation: numerical methods*. Prentice-Hall, 1989.

[57] Wikipedia contributors, "Danskin's theorem — Wikipedia, the free encyclopedia," 2022, [Online; accessed 6-June-2022]. [Online]. Available: https://en.wikipedia.org/w/index.php?title=Danskin%27s_theorem&oldid=1068534418

[58] H. Khalil, *Nonlinear Systems*. Pearson, 2001.

[59] E. Lavretsky, "Lasalle's invariance principle," 2022, [Online; accessed 6-June-2022]. [Online]. Available: http://www.cds.caltech.edu/archive/help/uploads/wiki/files/237/Lecture2_notes_CDS270.pdf

[60] S. H. Low and D. E. Lapsley, "Optimization flow control. I. basic algorithm and convergence," *IEEE/ACM Transactions on networking*, vol. 7, no. 6, pp. 861–874, 1999.