Imperial College London

PhD Thesis

Imperial College London

Dyson School of Design Engineering

On Distributed Ledger Technology for the Internet of Things: Design and Applications

Author:
Andrew Cullen

Supervisor:
Prof. Robert Shorten

December 21, 2022

Statement of Originality

I declare that the work presented in this thesis is my own, and that all contributions of others are appropriately acknowledged throughout. Specifically, joint authorships, collaborations and previously published work relevant to each chapter are clearly stated in Section 1.5.

Abstract

Distributed ledger technology (DLT) can used to store information in such a way that no individual or organisation can compromise its veracity, contrary to a traditional centralised ledger. This nascent technology has received a great deal of attention from both researchers and practitioners in recent years due to the vast array of open questions related to its design and the assortment novel applications it unlocks. In this thesis, we are especially interested in the design of DLTs suitable for application in the domain of the internet of things (IoT), where factors such as efficiency, performance and scalability are of paramount importance.

This work confronts the challenges of designing IoT-oriented distributed ledgers through analysis of ledger properties, development of design tools and the design of a number of core protocol components. We begin by introducing a class of DLTs whose data structures consist of directed acyclic graphs (DAGs) and which possess properties that make them particularly well suited to IoT applications. With a focus on the DAG structure, we then present analysis through mathematical modelling and simulations which provides new insights to the properties of this class of ledgers and allows us to propose novel security enhancements. Next, we shift our focus away from the DAG structure itself to another open problem for DAG-based distributed ledgers, that of access control. Specifically, we present a networking approach which removes the need for an expensive and inefficient mechanism known as Proof of Work, solving an open problem for IoT-oriented distributed ledgers. We then draw upon our analysis of the DAG structure to integrate and test our new access control with other core components of the DLT. Finally, we present a mechanism for orchestrating the interaction between users of a DLT and its operators, seeking to improves the usability of DLTs for IoT applications. In the appendix, we present two projects also carried out during this PhD which showcase applications of this technology in the IoT domain.

Acknowledgements

The research presented in this thesis was funded in part by a grant from IOTA Foundation and in part by a 2020 IBM PhD Fellowship Award[1]. I am grateful both to IOTA Foundation and to IBM Research for their support.

Despite the challenges and uncertainty that a PhD and a pandemic bring, my three years in London have been happier and more fulfilling than I could have hoped for. I attribute this to the relationships that I have formed/maintained during my time here.

Firstly, I am grateful to my supervisor, Prof. Robert Shorten for supporting and believing in me, for nurturing my creativity, and for always treating me and everyone around him with equal respect and kindness. I am also thankful for the support and guidance of Dr. Pietro Ferraro, especially during the early stages of my PhD. Working closely with Bob and Pietro has been formative for me, both academically and personally.

I had several fruitful collaborations with other researchers both from Imperial and other universities. I learned a great deal from Prof. Christopher King of Northeastern University, and from Roman Overko of University College Dublin. From Imperial, co-workers and friends who have enriched my experience include Lianna Zhao, Aida Manzano Kharman, Steve Kench, Liam Yassin and Cesare Caputo.

With the primary focus of my research revolving around distributed ledger technology, I had the opportunity to work closely with many brilliant scientists at IOTA Foundation. Some of the individuals who have contributed hugely to this thesis and to my own development include Bartosz Kuśmierz, Dr. Luigi Vigneri, Dr. William Sanders, Dr. Olivia Saa, Jonas Theis, Piotr Macek, Dr. Wolfgang Welz and Dr. Angelo Capossele.

I have also been fortunate enough to work closely with a number of exceptional researchers from IBM Research. Primarily, I would like to thank Dr. Sergiy Zhuk who mentored me as part of my IBM PhD Fellowship. I thoroughly enjoyed our weekly discussions and I am grateful to him for giving me autonomy to run with ideas and think deeply about problems. I am also thankful for the support of Dr. Mykhaylo Zayats and Dr. Jonathan Epperlein who shared their time and insights with me on numerous occasions.

Another activity which brought me great joy and satisfaction during my PhD was teaching. I would like to thanks Prof. Peter Cheung and Dr. David Boyle as well as Prof. Robert Shorten and Dr. Pietro Ferraro again, for entrusting me with teaching responsibility for their classes. I would also like to thank all the brilliant students from the Dyson School of Design Engineering for their enthusiastic engagement in all the modules I taught.

There are also a number of people who aren't quite sure what a distributed ledger is and were still indispensable to me during these three years. I thank my family for getting me here and making me who I am today. I thank Rob and Olly for their continued friendship and for giving me a solid base when we first moved to London. Finally, I thank my best friend and partner, Ciara, for always being there with a fresh perspective and a warm embrace.

[1] `https://research.ibm.com/university/awards/fellowships.html`

"It is as if the feel of thinking reveals something in the object itself, as if I were magically capable of grasping the ungraspable thing-in-itself, what in the thing is distinctly itself, not its data, its phenomena."

—**Timothy Morton**

Contents

List of Figures

List of Tables

Chapter 1

Introduction

1.1 Distributed Ledger Technology

Ledgers are record-keeping tools that play a vital role in our everyday lives. Consider, for example, the digital ledgers maintained by banks to facilitate and keep track of transactions between account holders—each time we make a purchase, our bank and the merchant's bank each update their ledgers to reflect the new balances in our accounts. We place our trust in the banks to maintain their ledgers securely to protect our funds and our privacy, and this in turn allows us to establish trust with other people. Ledgers enable us to transact safely not only with other humans but with machines, and they also support the reliable operation of fully automated machine to machine economies.

The ledgers maintained by banks and many other service providers are *centralised* in the sense that a single organisation has complete control over how the ledger is stored and updated. As such, any user of these ledgers must be willing not only to trust the managing organisation to behave honestly and maintain data securely but are often required to pay fees for the service provided, sometimes on a per-transaction basis. In light of recent scandals involving the unethical use of data by large organisations such as Facebook[2] and Google[3] as well as controversies regarding the high fees charged by ledger operators such as Visa[4], an alternative to centralised ledgers would be of great value. Distributed ledger technology (DLT) offers a promising solution.

A distributed ledger serves the same purpose as a traditional centralised ledger, but it is governed by more than one entity. In order for any changes to be made to the contents of a distributed ledger, agreement must be reached between these entities. Due to this decentralisation, appropriately designed DLTs prevent censorship, tampering and improper use of data by any individual or organisation. Additionally, a DLT presents a means for agents to transact in a trusted manner without paying fees to a central intermediary which unlocks a range of novel mechanisms in settings where connected computing devices gather and share data, i.e., the *internet of things* (IoT) [1].

1.2 Research Objective

The basic property required from a distributed ledger is that it should be a tamper-proof record of the data it stores. More specifically, data should be *immutable* and *irreversible* once it has been confirmed as part of the ledger. DLTs can also offer *transparency* because the ledger can often be viewed publicly whilst also preserving *privacy* because only cryptographic public keys of users are revealed on the ledger rather than their full identities. Other desirable properties for a distributed ledger may depend on the application, but for use in IoT settings the following are of particular importance to the design of a DLT.

[2] `https://www.bbc.com/news/technology-54722362`
[3] `https://www.nature.com/articles/d41586-019-03574-5`
[4] `https://www.bbc.com/news/business-54606252`

1. *Performance and efficiency:* the ledger should be able to efficiently process new updates at a high rate and with low latency/delay (the exact throughput and latency required depends on the specific application).

2. *Decentralisation and scalability:* control over the ledger state should be decentralised and distributed over a large number of entities.

3. *Fairness and security:* access to update the ledger state should be allocated fairly among the controlling entities and resilient against attacks by malevolent actors.

4. *Usability and fees:* individuals and organisations should be able to use the ledger easily and without paying fees or to pay fees for guaranteed quality of service.

The objective of this research is to advance the state of the art in distributed ledger design for IoT applications. In particular, we aim to analyse key aspects of existing DLTs which hinder their utility in the IoT domain with respect to the above properties. We then seek to find novel alternative approaches to DLT design which offer improvements with respect to the desirable properties listed above.

1.3 Background

DLTs consist of a ledger which is stored locally by *nodes* in a network, where each node or some subset can update the ledger locally and propagate these changes to neighbouring nodes. A node can be any computing device with sufficient resources and the ability to connect to a network, as depicted in Figure 1.1. In the context of IoT networks, edge devices with substantial computing and storage resources are the most likely candidates to operate as nodes, while smaller, more constrained devices may need to rely on these edge nodes to interact with the ledger on their behalf to avoid the often heavy burden of operating as a node. Nodes can apply updates to the ledger by appending a *block* to their local copy of the ledger and then broadcasting this update to neighbouring nodes. Blocks can include one or more transactions which change the balance of some accounts, for example, or may contain a collection of arbitrary data such as a sensor measurement from IoT devices. When nodes receive blocks from their neighbours, they must make decisions including whether it should be kept and whether it should be passed on to further neighbours.

Let us assume, for now, that each time a new block is issued by a node, they choose exactly *one* existing block to reference or *approve*, forming a chain of blocks which together comprise the ledger. This type of ledger is referred to as *blockchain*. When a node attaches a new block onto an existing chain of blocks, this signifies that the node agrees with the contents of these past updates to the ledger and that they wish to update the ledger with the contents of their new block. A node's local view of a blockchain is illustrated in Figure 1.2. The gray square represents the node's newly created block, while red squares represent blocks that do not yet appear to have received any approvals. These unapproved blocks are known as *tips*. Blue squares represent approved blocks, and green squares represent blocks that have received enough approvals to be deemed *confirmed*. The transparent blocks are not on the longest chain so may become *orphans*, as we shall explain in more detail below. For the remainder of this introductory section, we will focus on blockchains, but the terminology we introduce is applicable to a more general class of DLTs, some of which do not have a blockchain structure.

The precise details of how nodes append blocks to the ledger and disseminate them around the network vary between DLTs, but the key aspects of node behaviour that set DLTs apart from one another are as follows.

- *Confirmation:* the protocol used to decide when a block should be considered *confirmed* is a critical aspect of a DLT. A block should only ever be confirmed by a node if it is also eventually confirmed by all other nodes. Some DLTs provide only *probabilistic finality* for confirmed blocks, which means they will remain confirmed with high probability but there is some chance they will be reverted. Others provide *deterministic finality* meaning that a confirmed block is guaranteed to remain confirmed.

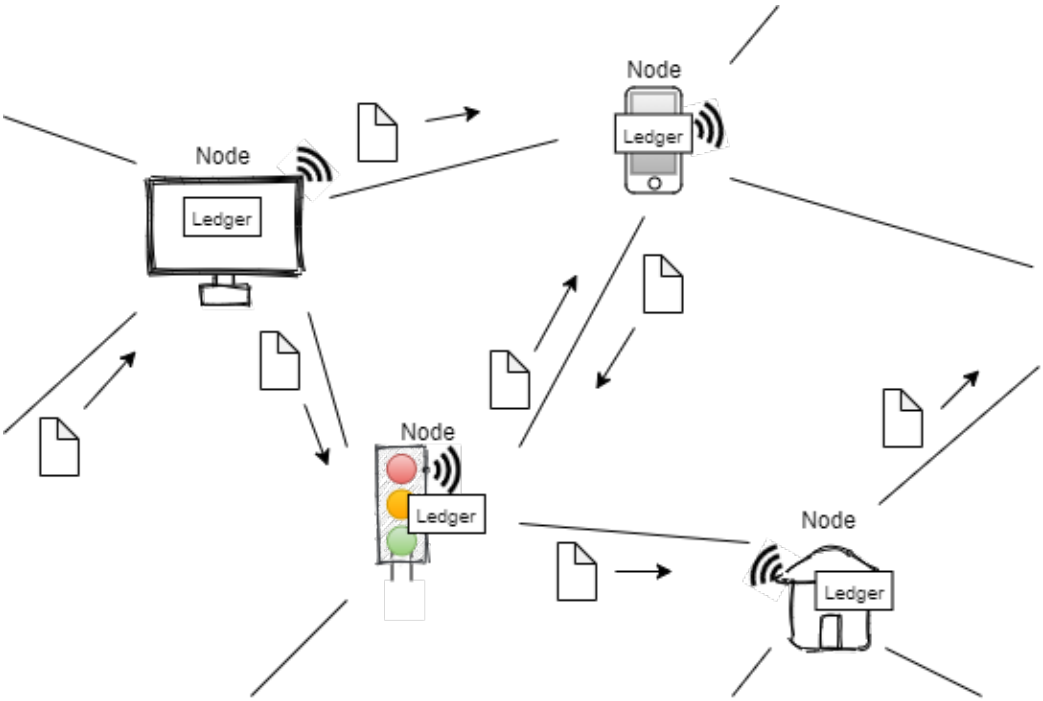


Figure 1.1: A DLT network with a variety of IoT devices participating as nodes. Nodes store a copy of the ledger and share ledger updates with neighbouring nodes.

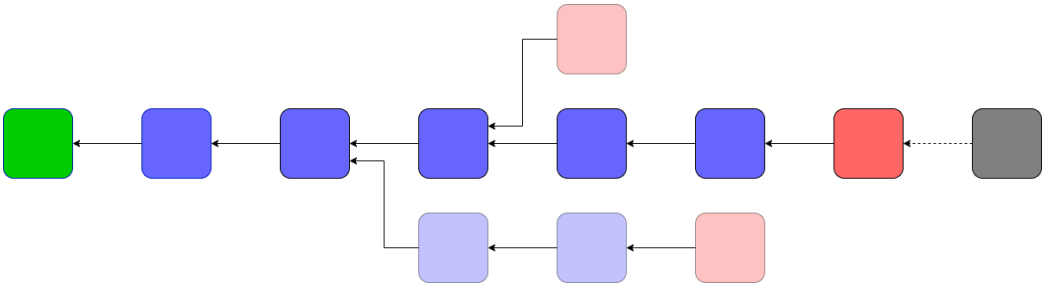


Figure 1.2: A node's local view of a blockchain ledger.

- *Tip selection:* the algorithm employed by a node to decide how to append a newly created block is another core component. New blocks are cryptographically linked to one or more existing blocks and attaching to a block signifies agreement with its contents and the contents of all blocks to which it points. In this way, attaching a block in a certain position signifies agreement with a particular portion of the ledger. A block attachment strategy is referred to as a *tip selection algorithm* (TSA). As we shall see, the TSA employed has a great influence on the entire structure of the ledger.

- *Access control:* the rules used to decide which blocks a node should process and forward on to neighbours are referred to as access control. Access control defines the governance of a DLT, prevents spamming and mitigates *Sybil attacks* in which an attacker attempts to gain an advantage by creating multiple identities.

We now delve into these three key aspects in further detail with a specific reference to blockchains, the simplest ledger structure.

1.3.1 Confirmation

When a node *confirms* a block, this means that they accept the data within as part of the ledger, and they can act on it. For example, any merchant should always wait for a customer's payment to be confirmed on the ledger before delivering any goods or services. Different DLT implementations will use different criteria to assess confidence in a block and decide when a block can be considered confirmed. There are generally two kinds of confirmation rule for blockchains: the first is referred to as *weight-based* confirmation; and the second is named *milestone-based* confirmation.

Weight-Based

The simplest weight-based confirmation rule is the k-block longest chain rule—a block can be considered confirmed if it is at least k blocks deep on the longest chain. The longest chain in Figure 1.2 is highlighted by the opaque blocks, while the blocks that are not on the longest chain are transparent. The reasoning behind this rule is that if a block has received a large number of approvals, it is highly likely that other nodes will continue to attach new blocks to this chain. In the Bitcoin blockchain [2], a 6-block longest chain rule is used to confirm blocks. We can think of each newly attached block as adding confidence or *weight* to the blocks it approves directly or indirectly. We can then define the *cumulative weight* of a block i as the sum of the weights of all blocks that approve i either directly or indirectly. Another way of stating the k-block rule is then to say that a block must be on the heaviest chain and have cumulative weight of at least k to be considered confirmed.

Weight-based confirmation gives *probabilistic finality* meaning that for sufficiently large k, it is highly unlikely that a longer/heavier chain will appear and replace the confirmed chain. Other weight-based rules attempt to include more than one chain so that the other chains (the transparent blocks in Figure 1.2) do not go to waste. We discuss other weight-based approaches in Section 1.4.

Milestone-Based

Milestone-based confirmation does not rely directly on the notion of cumulative weight. Instead, we introduce special blocks called *milestones* which have been agreed upon in a more concrete way than a normal block, and everything that is approved directly or indirectly by a milestone can be considered immediately confirmed. Milestones may be issued by a central authority to provide security for a DLT in the early stages of adoption when a reputation is not yet well distributed across nodes[5]. Milestones may also be issued by elected leaders or may alternatively be issued via multi-signatures (see Appendix C and [3]) by a consortium of high reputation nodes. The latter approach requires Byzantine fault tolerant (BFT) agreement on where to attach each milestone so milestones can only be issued by consortiums in this way if the group of signers is reasonably small and the interval between milestones is reasonably long. This is due to the fact that while BFT algorithms can cope with nodes that deviate arbitrarily from the protocol (Byzantine nodes), they do not scale well to large groups of participating nodes.

Milestone-based confirmation provides *deterministic finality*. Some DLTs with a small and tightly controlled group of governing nodes may have every block as a milestone so they can provide immediate and irreversible finality from every block. However, this approach does not scale well to large numbers of nodes, so it comes at the cost of centralisation. Other DLTs may use a combination of weight-based and milestone-based confirmation, where milestones are issued at regular intervals but not for every block.

1.3.2 Tip Selection

The algorithm used to select which existing block to approve with a newly created block is known as a *tip selection algorithm* (TSA). As discussed above, appending a block to a particular chain directly contributes to the blocks in that chain becoming confirmed, so it can be thought of as casting a vote on this part of the ledger. As such, the decision rules for where to attach new blocks require careful consideration. The most common TSA for blockchains is the longest chain rule, as employed in the Bitcoin blockchain, wherein new blocks are attached to the tip of the longest

[5] https://blog.iota.org/coordinator-part-1-the-path-to-coordicide-ee4148a8db08/

chain visible to the node creating the block. The idea of the longest chain rule is simply to vote on the version of the ledger which the majority of other nodes have voted on, so if all nodes see the same blocks, they should all attach their blocks to the same chain and contribute to its eventual confirmation.

Note that if more than one node attempts to attach a block to a blockchain at the same moment, only one of these blocks can ultimately make it on to the longest chain and become part of the ledger. This fact severely limits the throughput of blockchains and result in many blocks (the transparent blocks in Figure 1.2) never being confirmed, hence wasting resources. If the size of blocks is increased and/or the interval between them decreased in an attempt to improve throughput, block propagation delays eventually result in forking of the blockchain and many blocks never being confirmed. Another path to improving throughput without wasting blocks is to allow each new block to select more than one tip for approval. This results in a *directed acyclic graph* (DAG) structure instead of a chain. The various classes of tip selection algorithm for DAGs are more complex than for blockchains so we defer further discussion to Chapter 2.

1.3.3 Access Control

Distributed ledgers are not maintained by any single actor, and as such, access to modify the ledger must be carefully controlled by predefined rules—an *access control* mechanism is required. Due to the fact that attaching a new block constitutes a vote, the rate at which a node can write new blocks directly determines their power over the network, which makes access control vital to the security of any distributed ledger. A naive implementation of a public DLT without access control would be to assign a fixed block rate to each digital identity. This implementation would be vulnerable to *Sybil attacks* in which a malicious actor creates numerous identities to gain additional control over the ledger. Access control is sometimes referred to as *Sybil protection* because it prevents these kind of attacks by imposing objective criteria for writing new blocks. Access control types include Proof of Work (PoW), Proof of Stake (PoS), permissioned systems and Proof of Reputation (PoR), which generalises PoS, permissioned systems and delegated PoS.

Proof of Work

PoW is the access control type employed in Bitcoin [2] and many of the world's most prominent DLTs at the time of writing. PoW access control requires nodes to solve a computationally challenging puzzle, namely inverting a hash function, to write a block to the ledger. Inclusion of a solution to the PoW puzzle in a block proves that energy has been consumed to create it and if a node wished to write blocks at a greater rate, it has no option but to acquire more computing power and consume it. Broadly speaking, the majority holders of computing power control the state of the ledger in DLTs with PoW access control. PoW has been popular to date due to its robustness and the simplicity of implementing it in blockchain or DAG-based ledgers. However, PoW remains unsuitable for many DLT applications due to issues such as the enormous amount of power consumed to maintain such a ledger. PoW also creates a barrier to participation for nodes with limited computing resources such as IoT devices, so alternative access control methods are of great importance for IoT applications of DLT. PoW ledgers rely on weight-based confirmation which only offers probabilistic finality.

Proof of Stake

PoS access control offers an appealing alternative to PoW, eliminating the latter's need for wasteful energy consumption. PoS assigns nodes access to write blocks at a rate proportional to their wealth in the native ledger currency. Implementation of PoS is generally not as straightforward as PoW, but a number of solutions have been proposed for blockchains. One method of implementing PoS access control is to use a decentralised random beacon to elect a stakeholder node at random to issue each block with probability of election proportional to their stake [4]. Such protocols can also be modified to remove the need for a random beacon [3]. Weight-based confirmation can be used in these cases, or alternatively, milestone-based confirmation can be achieved with the use of BFT algorithms among a committee of stakeholders to agree on blocks at fixed intervals [5]. BFT-based methods do not scale well to large numbers of nodes so a sub-committee must be elected to perform the BFT protocol [6]. No methods for incorporating PoS access control into DAG-based ledgers

had been proposed prior to [7, 8] which is presented in Chapter 4 of this thesis.

Permissioned Ledgers

Permissioned ledgers are those in which the participating nodes and their voting power is defined from the outset, as opposed to a public ledger in which any node can join the network and participate by simply acquiring some resource such as computing power (PoW) or some coins (PoS). Permissioned ledgers have an element of centralised control in the form of the individual or committee that decides the initial distribution of voting rights [9]. These ledgers are popular in enterprise settings where an organisation of group of organisations wish to share a ledger, so the consensus group (nodes with voting rights) is relatively small. As a result, permissioned ledgers can employ milestone-based confirmation to achieve excellent performance and deterministic finality, albeit at the cost of centralisation [10].

Proof of Reputation

Definition 1.3.1 (Reputation)**.** Reputation is a quantity associated with each node's identity which is difficult to obtain and on which all nodes have consensus.

Examples of reputation include stake (as in PoS), delegated stake (dPoS) such as IOTA's mana system, and externally managed access rights as in permissioned ledgers. We use the term *reputation* in this work because the access control presented in Chapter 4, initially designed for IOTA's mana system, is more broadly applicable and PoR captures its generality. In other words, by designing an access control mechanism for PoR, it can be readily applied to any PoS, dPoS or permissioned system.

1.4 Related Research

This thesis covers a wide range of problems all of which relate to the design and application of distributed ledgers. Each chapter draws on previous research from different areas to find novel solutions to emerging problems related to DLT. As such, we reference the literature most relevant to each chapter therein, and for now we simply give an overview of other DLT research as it relates to the background we have provided so far. It is worth noting that this section is not intended to serve as a complete survey of the state-of-the-art in DLT research—numerous papers already serve this purpose and to repeat their efforts would be of little value [11, 12, 13, 14].

The Bitcoin blockchain [2] represents the first public DLT and still holds the highest market value at the time of writing this thesis. Bitcoin employs PoW for access control, the longest chain rule for tip selection and a 6-block rule for confirmation. A maximum block size of 1MB is also specified in the protocol which limits the number of transactions per block, and the difficulty of the PoW is adapted so that new blocks can be created every ten minutes, on average. These two parameters (block size and interval) determine the theoretical maximum throughput of a blockchain, which in the case of Bitcoin is around 7 transactions per second (TPS), and with a 6-block rule, confirmation takes over an hour. The reasonably small block size and long interval used in Bitcoin makes the network highly robust and resilient against poor network connectivity and long delays but severeley limits its performance in terms of TPS and confirmation times [15]. Many other cryptocurrency market leaders are derivatives of Bitcoin with changes made to these key parameters and other features such as the hashing algorithm used. For example, [16, 17, 18] are simply variants of this traditional PoW model, whilst others are merely tethered to or hosted on existing blockchains [19, 20]. One approach to improving the scalability of these PoW blockchains are so-called *layer 2* solutions which operate outside of the core DLT protocol and periodically reference the blockchain for security [21, 22].

Another avenue of research towards improving upon the aforementioned blockchain architectures is focussed on novel ledger structures. Specifically, by using different confirmation rules and tip selection algorithms we end up with more complex ledger structures than a simple blockchain. An early proposal for a more flexible structure than a blockchain was the Greedy Heaviest Observed Sub-Tree (GHOST) confirmation rule [23] proposed to modify the bitcoin longest chain rule. The idea behind GHOST, a version of which is employed in the Ethereum blockchain [24] is to allow

multiple branches of a tree to be confirmed rather than a single chain, provided there are no conflicts between branches. This reduces the number of orphan branches and permits higher throughput (larger blocks and shorter block intervals) whilst retaining security properties in the presence of long network delays. This concept of allowing more branches to be confirmed can be extended into tip selection, i.e., each new block can be allowed to approve two or more existing blocks. This gives rise to the DAG structure [25]. A number of distributed ledgers now employ DAG structures, including IOTA [26], Hedera [27], Meshcash [28] and Byteball [29]. Other ledgers use more restrictive DAG-based structures such as that of Nano [30] which comprises a separate chain for each individual user account. Numerous experimental and theoretical studies of DAG-structured ledgers have appeared in recent years [31, 32, 33, 34, 35, 36, 37, 13].

An alternative research direction aimed at improving upon early blockchains is based around alternative access control to replace PoW. PoW is highly wasteful of energy [38] and presents an entry barrier for contribution to a DLT network because specialised hardware must be acquired to participate. PoS, as introduced in 1.3.3 is a promising alternative which has received a great deal of attention in recent years. A number of PoS blockchains have entered the cryptocurrency market such as Algorand [4] and Cardano [39], and the Ethereum blockchain is making a transition to PoS from PoW [40] at the time of writing. These PoS blockchain solutions all consist of sequential leader elections which are made in a distributed manner, and consensus is based on rules such as the longest chain rule [2] or GHOST [23] just as in PoW blockchains. Decentralised random number generators (dRNGs) are the core component of any such protocol. We explain the role of dRNGs and review the state of the art in Appendix C. Other approaches to implementing PoS access control in blockchains are based on combination of committee selection and BFT voting algorithms for milestone-based confirmation rather than weight-based confirmation [41, 42]. An excellent survey of PoS blockchain solutions with a focus on attacks and security vulnerabilities is presented in [43]. It is worth noting once again that the fundamental operation of any of these PoS approaches can be extended to more general PoR by using any other uninflatable quantity, which we refer to as reputation, in the place of stake. The design of a robust reputation systems or system for delegating voting rights to other individuals (delegated PoS) can generally be treated as a separate problem [44]. PoS access control can not be applied to DAG-based ledgers with the techniques discussed above and as such, no PoS DAGs have been proposed prior to [8] which is the subject of Chapter 4 of the present thesis.

Both PoW and PoR (including PoS) are prone to centralisation for a variety of reasons. Any unregulated scarce resource is subject to economic factors that can lead to accumulation and result in unequal distribution [45]. Permissioned distributed ledgers offer a predefined and accepted degree of centralisation which can greatly simplify almost every aspect of their design. When a sufficiently small committee of trusted nodes are responsible for consensus over the ledger, BFT voting algorithms can be used to rapidly reach agreement on every block and confirm transactions with latency comparable to centralised systems [10]. Permissioned blockchains such as Hyperledger [9] are popular in enterprise settings for these reasons. Voting algorithms are also increasingly being used in public distributed ledgers to provide final confirmation of transactions [46, 41, 47]. A key challenge for incorporating such voting algorithms into public DAG-based ledgers is that of committee selection [48].

While DLT is best known for its use in cryptocurrencies, many novel applications areas in the realm of the IoT have also been actively studied as the technology develops. Supply chain management is one such area where DLT offers clear advantages for its ability to establish trust and cut out intermediaries to improve the efficiency of supply chains [49]. Electronic health provides another example of an area where DLT can be a powerful aid because of its ability to provide both transparency and privacy-preserving properties (users do not need to reveal their identities in order to write information to a public distributed ledger). [50] discusses how DLTs can be used to ensure integrity and authenticity of health-related data collected from embedded and wearable IoT devices. A broad range of challenges related to applying DLTs in a healthcare setting are addressed in [14], ranging from verifying credentials of medical practitioners and managing clinical trials to designing privacy-preserving machine learning tools with health data. DLT has also been proposed as a tool to combat Covid-19 with the help of various medical IoT devices [51]. In [1], DLT is proposed to enforce compliance with social contracts and actuate agents in emerging problems from the domain of smart cities, and in [52], DLT tokens are used to support efficient and private sampling

for routing problems. Further applications of DLT to smart mobility problems are discussed in Appendix B.

1.5 Thesis Structure, Contributions and Collaborations

1.5.1 Chapter 2

In Chapter 2, we describe DAG-based ledgers in detail as a generalisation of the blockchains detailed above. We then present a fluid model for a DAG-based ledger and predict the properties of the ledger under various delay models. The models and accompanying simulations in this chapter contribute new insights to the behaviour of DAG-based ledgers under varying network conditions which can be used to guide the design of core components of these protocols.

The work on variable delay models was conducted in collaboration with Prof. Christopher King of Northeastern University and Dr. Pietro Ferraro and Prof. Robert Shorten of Imperial College London. Prof. King provided the fluid models while Dr. Ferraro and Prof. Shorten formulated the discrete model from which the fluid model is derived [1]. The present author designed test scenarios and implemented all simulations and tests.

1.5.2 Chapter 3

In Chapter 3, we continue our analysis of DAG-based ledgers by considering an attack known as a parasite chain attack. We analyse the security of a DAG-based ledger under a biased random walk tip selection algorithm, and we then present modifications which further improve security. The analysis performed in this chapter contributes new knowledge of how parasite chain attacks can be prevented through appropriate choice of protocol parameters as well as proposing an improvement to state of the art tip selection algorithms.

This chapter contains joint work with Prof. Christopher King of Northeastern University and Dr. Pietro Ferraro and Prof. Robert Shorten of Imperial College London. All three of these collaborators contributed to the formulation of the simple parasite chain model and design of testing scenarios, Dr. Ferraro proposed the model for the random walk tip selection based on absorbing Markov chains. The present author played a central role in all of the above and additionally coded and executed the simulation scenarios and presented results and insights.

1.5.3 Chapter 4

In Chapter 4, following on from our analysis of the DAG-structured ledger, we address the problem of access control for DAG-based DLTs. Specifically, we present a new access control algorithm which removes the need for PoW in DAG-based DLTs for the first time. The removal of the need for PoW in DAG-based DLTs represents a marked improvement in the state of the art. In particular, this contribution serves to dramatically improve the efficency of this class of DLTs, and the analysis provided here suggests that other desirable properties of DLTs for IoT applications are also retained by this approach.

This chapter contains joint work with Dr. Luigi Vigneri and Dr. William Sanders of IOTA Foundation and Dr. Pietro Ferraro and Prof. Robert Shorten of Imperial College. Dr. Vigneri and Dr. Sanders provided an initial problem statement, driven by a need from IOTA Foundation to remove PoW from their IoT-oriented DLT. All four of these collaborators then contributed parts of this solution over the course of our collaboration. The present author designed the key components of the solution such as the scheduler and implemented all simulations and analysis presented here.

1.5.4 Chapter 5

In Chapter 5, we present an extended form of the core access control algorithm which additionally takes the DAG structure of the ledger into account and incorporates tip selection directly into the data flow. Chapter 5 builds on all prior chapters and contributes a deeper understanding of how access control and the DAG-structured ledger are linked. The chapter also provides the first

integration of access control into a more complete DLT protocol and demonstrates its impact on other modules including consensus.

Contributors to the development of the access control improvements in this chapter include numerous members of the IOTA Foundation's networking team, including Piotr Macek, Dr. Olivia Saa, Dr. William Sanders, Jonas Theis, Dr. Luigi Vigneri and Dr. Wolfgang Welz. The present author played a leading role in these design improvements and is responsible for all implementations and testing presented here.

1.5.5 Chapter 6

In Chapter 6 we present a user-node interaction mechanism which addresses another vital aspect of DLT for IoT applications, namely usability. Whilst mechanisms of this kind are well studied for blockchains, the solution presented in Chapter 6 is the first to address this problem for DAG-based DLTs. This work represents an important step towards making DAG-based ledgers more practically usable in a broader range of settings.

Development of this mechanism was joint work with Dr. Luigi Vigneri of IOTA Foundation and Lianna Zhao and Prof. Robert Shorten of Imperial College London. Simulations were carried out by Lianna Zhao under the guidance of the present author by building code on top of the simulator used to produce the results in the two preceding chapters. The present author played a leading role in the design of the solution and the experiments carried out.

1.5.6 Publications

Some of the work presented in this thesis has also been included in the following publications:

- A. Cullen, P. Ferraro, C. King, and R. Shorten, "Distributed ledger technology for smart mobility: Variable delay models," in *2019 IEEE 58th Conference on Decision and Control (CDC)*, 2019, pp. 8447–8452.
- A. Cullen, P. Ferraro, C. King, and R. Shorten, "On the resilience of DAG-based distributed ledgers in IoT applications," *IEEE Internet of Things Journal*, vol. 7, no. 8, pp. 7112-7122, 2020.
- A. Cullen, P. Ferraro, R. Shorten, W. Sanders, and L. Vigneri, "Access control in adversarial environments for IoT-oriented distributed ledgers," in *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*, 2021, pp. 968-973.
- A. Cullen, P. Ferraro, R. Shorten, W. Sanders, and L. Vigneri, "Access control for distributed ledgers in the internet of things: A networking approach," *IEEE Internet of Things Journal*, vol. 9, no. 3, pp. 2277–2292, 2022.
- L. Zhao, L. Vigneri, A. Cullen, W. Sanders, P. Ferraro, and R. Shorten, "Secure access control for DAG-based distributed ledgers," *IEEE Internet of Things Journal*, vol. 9, no. 13, pp. 10792–10806, 2022.
- A. Cullen, L. Zhao, L. Vigneri, and R. Shorten, "Improving quality of service for users of DAG-based distributed ledgers," 2022. Available: `https://arxiv.org/pdf/2203.12076.pdf`.

Other outputs produced during this PhD related to the application of DLT in IoT settings which have not been included in the main body of this thesis (see Appendix A and B) are presented in the following publications:

- A. Cullen, P. Ferraro, G. Russo, and R. Shorten, "Ad-hocChain: Cooperative sharing and trading infrastructure for electric vehicle charging networks," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, 2019, pp. 207-212.
- R. Overko, R. Ordónez-Hurtado, S. Zhuk, P. Ferraro, A. Cullen, and R. Shorten, "Spatial

positioning token (SPToken) for smart mobility," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 2, 2022.

Chapter 2

DAG-based Distributed Ledgers and Variable Delay Models

***Abstract*— Early distributed ledger technologies exclusively adopted blockchain structures. This simple structure requires any updates to the ledger to be added sequentially, so any concurrent attempts to add a block to the ledger will result in only one of these blocks being confirmed. Directed acyclic graphs (DAGs) represent a generalisation of the chain structure which allows blocks to be added in parallel and hence removes the limitations placed on blockchains. In this chapter we give an overview of how DAG-based ledgers operate, extending the DLT concepts introduced in Chapter 1 to cover DAGs. We then present a model for the evolution of a DAG-based ledger in a network with variable delays, building on previous modelling work for DAG-based DLTs. Analytical expressions are derived, and simulation results are provided to show how the DAG evolves as a dynamical system. Part of the work on variable delay models presented in this chapter also appears in [53].**

In this chapter, we introduce a class of distributed ledgers with more flexible structures than blockchains, namely DAGs. DAG-based ledgers are promising alternatives for IoT applications, as we can highlight with reference to the design principles outlined in Chapter 1: high throughput, low latency, decentralisation, usability and fees, security, and fairness. Blockchains are fundamentally limited in their performance because blocks must be added in sequence and sufficient time must be left between blocks to allow for network delays. DAG-based DLTs allow blocks to be added to the ledger in parallel which removes the limitations of the blockchain structure and permits *high throughput* and *low latency*. Due to the competitive nature of issuing blocks in a blockchain, nodes are incentivised to pool their resources[6] and it is not feasible for less powerful actors to participate as nodes and issue blocks. In DAG-based ledgers, on the other hand, the ability to issue blocks in parallel reduces competition between nodes to issue blocks because more than one concurrent block can be confirmed. As a result, less powerful actors can participate as nodes and issue blocks which leads to improved *decentralisation* over blockchains. When users can easily operate as nodes and issue their own blocks in this way, this also allows them to avoid paying *fees* to other nodes as is required in most blockchains [54]. DAG-based ledgers can be designed to have good *security* and *fairness* properties as we shall show in later chapters when we present the design of specific tip selection and access control algorithms.

2.1 Directed Acyclic Graphs

A sequence illustrating blocks being issued in a DAG is shown in Figure 2.1. Similarly to Figure 1.2, the grey square represents a newly created block, red squares represent tips, blue squares represent approved blocks and green are confirmed. Suppose we have one block A which is approved directly

[6] `https://btc.com/stats/pool`

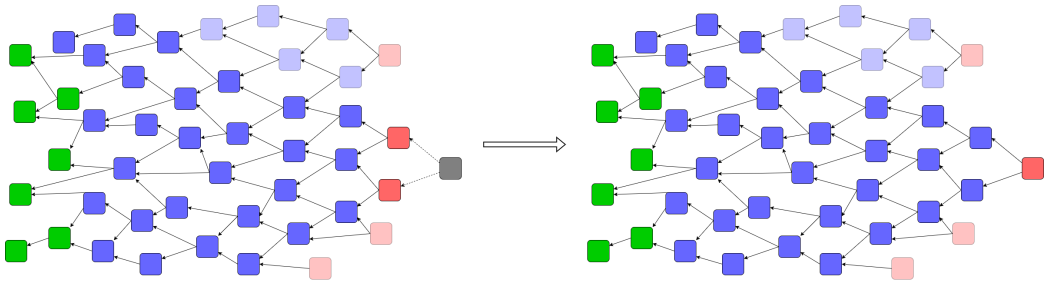


Figure 2.1: Sequence to issue a block in a directed acyclic graph. On the left, we see the first part of the sequence in which a new grey block arrives and selects two red tips. On the right, we see that once the grey block is added to the DAG, it becomes a tip and the two blocks it approves cease to be tips.

by another block B: we then refer to A as B's parent and we refer to B as A's child. The sequence of Figure 2.1 shows how the newly issued blocks become tips and the blocks that they approve cease to be tips once they become visible to other nodes. The opaque blocks are those that are directly or indirectly approved by the new grey block, and we refer to this as the *past cone* of the grey block. Similarly, the term *future cone* can be used to refer to all blocks that directly or indirectly approve a block. All blocks in a new block's past cone must be consistent with one another for the new block to be deemed valid, for example, if blocks contain transactions they must all be consistent with one another and not spend the same tokens more than once. The topic of double spending is discussed in detail in Chapter 3. In general, a new block in a DAG-based ledger can approve two or more tips, but for the purpose of our analysis we will always assume that each block approves exactly two existing blocks. As in the previous chapter when we introduced blockchains, we can discuss the key features of a DAG-based ledger with reference to the three categories: confirmation, tip selection and access control.

2.1.1 Confirmation

Confirmation in DAG-based ledgers follows similar principles as for blockchains, and we can also identify the same two main types of confirmation for DAGs, namely weight-based and milestone-based confirmation.

Weight-Based

Each time a new block is appended to the DAG, the cumulative weight of all blocks in the new block's past cone are increased by the weight of the new block. Figure 2.2 shows an example of how cumulative weight changes in time, where the weight of each block is assumed to be exactly one. Weight-based confirmation rules can be generalised to DAG-based ledgers using the notion of cumulative weight. The k-block rule for blockchains, for example, generalises easily to DAGs—nodes should wait until a block has cumulative weight of at least k to consider it confirmed.

Milestone-Based

Similarly, milestone-based confirmation can be applied to DAGs—every block in the past-cone of a milestone can be considered confirmed, as illustrated in Figure 2.3 where the yellow block represents a milestone. Just as in blockchains, milestones may be issued by a single central entity, although this essentially results in a completely centralised ledger and many of the benefits of decentralisation are lost. Committees can also be elected and BFT voting algorithms used to agree on milestones among a small subset of powerful nodes.

2.1.2 Tip Selection

While simple tip selection policies such as the longest chain rule are sufficient in blockchains, the equivalent policies for a DAG require more careful consideration due to the added complexity of the ledger structure. However, the principle behind tip selection is the same for DAGs as blockchains

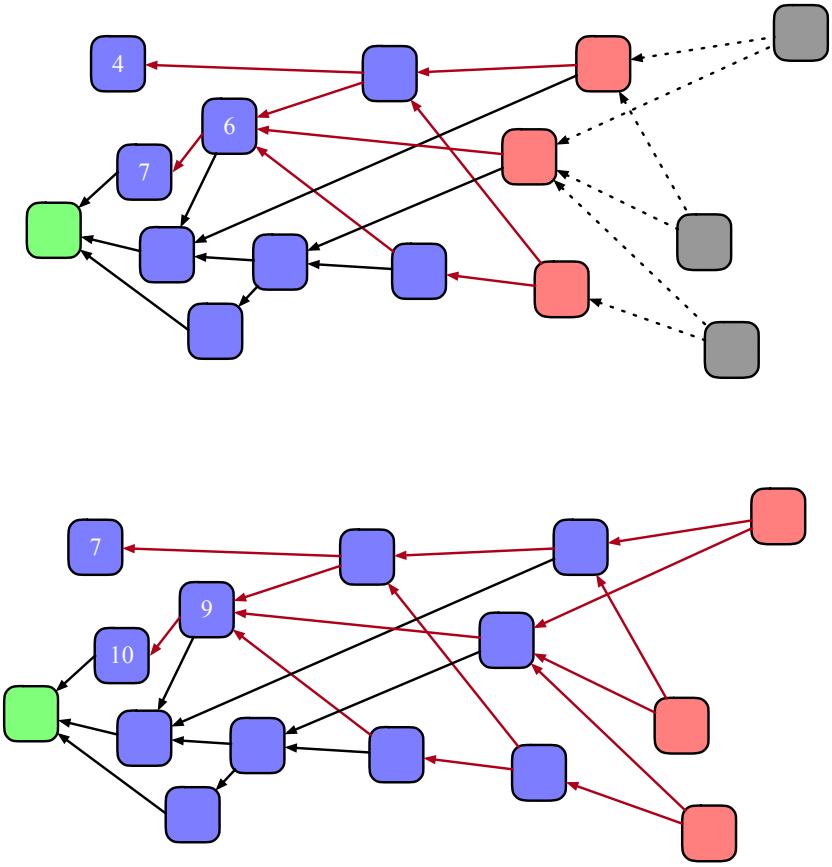


Figure 2.2: Evolution of the cumulative weight of three blocks as three new blocks enter the DAG.

in the sense that a node attaching a new block can be thought of as casting its vote on the part of the DAG that this node believes to be correct and consistent with the majority of other nodes. The security of DAG-based ledgers depends heavily on how tips are selected by new blocks, as we shall discuss in Chapter 3. An overview of TSAs that have been proposed for DAGs can be given as follows.

Uniform Random Tip Selection

Uniform random tip selection (URTS) algorithms select two tips randomly from the set of all tips or from a subset of these tips. URTS in its basic form, where all tips are considered eligible, can be vulnerable to attack as it does not give any sense of which tips correspond to good or bad parts of the DAG. URTS can be made secure by considering only the subset that meet some locally measured and objective criteria such as time since block creation. This can help to ensure that honest nodes only attach their blocks to good parts of the DAG rather. The upper panel of Figure 2.4 shows an example of the basic URTS procedure. The interested reader can refer to [26] for a detailed discussion on this topic.

Biased Random Walk

Biased Random Walk (BRW) algorithms (also referred to as Monte Carlo Markov chain (MCMC) tip selection algorithms [26]) involve creating two independent random walks in the interior of the DAG, as illustrated in the lower panel of Figure 2.4. Note that while the two random walks displayed in this figure are disjoint, this need not be the case, however, if the second random walk

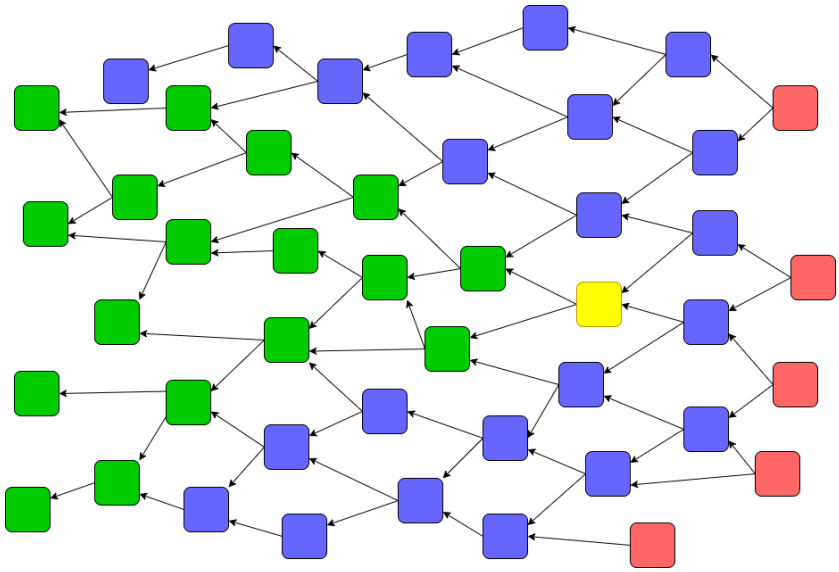


Figure 2.3: Milestone-based confirmation—all blocks in the past cone of the yellow milestone block are immediately confirmed.

terminates at the same tip as the first then it must be repeated. The walk may start at the very first block (known as the *genesis* block) or somewhere deep in the DAG and move along the edges of the graph. The start point of the random walk can be chosen, for example, to be any block with cumulative weight above a chosen threshold, or could be selected to be some past milestone (see Section 2.1.1). When a new node joins the network, they must *bootstrap* from some neighbouring nodes and passively observe the growth of the DAG for some time before performing BRW tip selection because they must know of all the blocks in the future cone of the walk's start point. The probability of jumping along an edge from block j to block k is proportional to $f(-\alpha(\mathcal{H}_j - \mathcal{H}_k))$, where $f(\cdot)$ is a monotonic increasing function (generally an exponential), α is a positive constant and $\mathcal{H}_i$ represents the cumulative weight of a block i. The walk stops when it reaches a tip, which is then selected for approval.

Hybrid Approaches

Due to the fact that two independent tip selections must be made for the attachment of each new block, hybrid approaches can be employed which combine a URTS and a BRW selection in an attempt to capture the good properties of each [31].

URTS algorithms are cheap and simple to implement, which is of high importance for DLTs to prevent barriers to network participation. Simple rules for selection of a subset of tips can make these simple algorithms effective and secure. BRW algorithms, on the other hand, can be expensive to implement due to the computational cost of random walks on DAGs. However, BRW algorithms offer better security properties and are the closest analogue of the longest chain rule for blockchains—they are more likely to select tips of heavier branches in a DAG which can be thought of as voting for the branch of the DAG which some majority has voted for. For the remainder this chapter, we restrict our attention to URTS to simplify our analysis, and we revisit BRW algorithms in Chapter 3 where we explore their security properties.

2.1.3 Access Control

As mentioned in Chapter 1, PoW access control can be easily applied to DAG-based ledgers—nodes must solve a computationally difficult puzzle and include the solution for each block they issue. This essentially allocates a block issuing rate to each node proportional to their expended computing power. However, PoR access control such as PoS cannot be generalised so easily to DAG-based ledgers. PoR implementation for blockchain involve leader elections in which a node

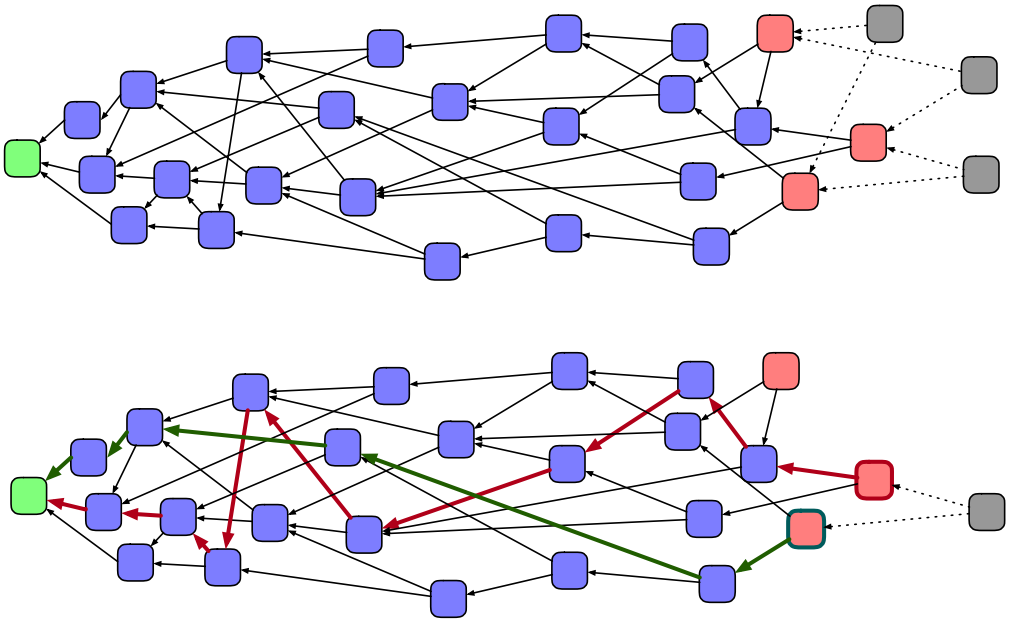


Figure 2.4: Two main classes tip selection algorithms: URTS (above) and BRW (below). URTS simply selects two tips at random from the tip set, while BRW uses a random walk from deep in the DAG. The red and green lines represent two independent random walks.

is elected at each round to issue a block, where the leader node is selected using some trustworthy source of randomness. These approaches are inherently sequential, relying on the existence of some fixed *rounds*, so they do not extend to DAG-based ledgers in a straightforward way because blocks do not have a total ordering in DAGs. The first PoR access control for DAG-based distributed ledgers is the subject of Chapter 4 of this thesis.

2.2 Variable Delay Models

We now present a more precise model for a DAG-based distributed ledger as a dynamical system to gain a better understanding of how the DAG structure evolves. In a real DLT network, there would be multiple local copies of the DAG and each user would independently update its own copy, but for simplicity of our analysis we consider only one such local copy. We assume that all nodes access this single DAG, but new blocks experience a delay before appearing in the ledger. We assume that each new block selects two tips for approval at random from the set of all visible tips (the URTS algorithm) and attempts to validate them. The DAG, $G(t)$, contains the record of all blocks which arrived at or before time t. If validation fails, the choices are discarded, and another two tips are selected for validation. This continues until the process is successful, and we assume that this whole validation effort is essentially instantaneous. However, after the validation there is a waiting period, H, before the new block becomes eligible to be selected as a tip by subsequent blocks. This delay seeks to model processes such as finding a PoW solution, propagating blocks to neighbouring nodes, or any other time-consuming processes required by a particular DLT implementation.

It is important to note that the tips selected by a new block remain as tips during this delay period, so they may be selected by one or more other new blocks. After the delay period, the new block appears in the ledger, and the two parent blocks cease to be tips and are no longer available for selection by other new blocks (at least, by the ones that follow the protocol). In the remainder of this section, we assume that the delays for new blocks are random and independent, with some fixed distribution. Let $\mathcal{L}(t)$ denote the set of tips at time t. Then we assume that when a new

block arrives at time t, it has selected two tips at random from the set $\mathcal{L}(t-H)$ (where H is the random delay time). Thus, there are two random elements in the algorithm: the random delay time H; and the random selection of two tips from the tip set at the earlier time. For simplicity of analysis, we assume that URTS is used here (see Section 2.1.2).

Note that the work presented in this section builds upon earlier research into modelling DAG-based distributed ledgers as dynamical systems [1] where delays were assumed to be constant. The model presented here assumes the delay of each block to be a random variable and solutions are given for a number of special cases of its distribution. Since the publication of this work [53], other research has emerged which takes a different approach to studying variable delay models of DAG-based DLTs. [36] assumes the delay of each block belongs to one of discrete set of delay classes, and similar results are presented to model the behaviour of DAG-based DLTs.

2.2.1 Tip Selection Probability

Let $\{T_n\}$ denote the increasing sequence of times when new tips are added to the DAG so that

$$0 \leq T_1 \leq T_2 \leq \cdots \leq T_j \leq \cdots . \tag{2.1}$$

We will label a block by the time when it was added to the DAG. Thus block j was added to the DAG at time T_j, and remains in the tip set until some future time when it is approved. For $j < n$ we define $a_j(T_n)$ to be the indicator variable for the event that block j is still a tip at time T_n.

$$\begin{aligned} a_j(T_n) &= \begin{cases} 1 & \text{if } j \in \mathcal{L}(T_n) \\ 0 & \text{otherwise} \end{cases} \\ a_n(T_n) &= 1 \end{aligned}$$

Note that $a_j(T_n) \geq a_j(T_m)$ for all $n \leq m$. Also, defining $L(t)$ as the number of tips at time t, we have

$$L(T_n) = \sum_{j=1}^{n} a_j(T_n) \tag{2.2}$$

Consider now the arrival of a new block at time T_n. This new block must select two tips for validation from the set $\mathcal{L}(T_n - H)$, where H is the random delay time. We define $\tau(T_n)$ to be the set of two blocks which are selected for validation by block n (i.e., at time T_n). Suppose that block j is a tip at time T_n, and that the random delay time H satisfies $H \leq T_n - T_j$. Since $T_n - H \geq T_j$, this means that the block j had already been added at time $T_n - H$, and since it is assumed to still be a tip at time T_n, it must also be in the tip set $\mathcal{L}(T_n - H)$. Thus the probability that block j is selected for validation at time T_n is simply the probability that any tip is selected for validation out of all the tips in $\mathcal{L}(T_n - H)$, which is

$$p(T_n - H) = \frac{2}{L(T_n - H)} - \frac{1}{L(T_n - H)^2} \tag{2.3}$$

(the second term in (2.3) accounts for the fact that the same tip can be chosen twice by the URTS algorithm). This result can be formalised in the following way. We define $\mathcal{F}(n)$ to be the σ-algebra generated by the DAG up to time T_n. Thus by conditioning on $\mathcal{F}(n)$ we are fixing the history of the DAG, including of course the tip sets at all previous times. Also note that $a_j(T_n) = 1$ if and only if the block j is a tip at time T_n. Thus

$$\begin{aligned} &\mathbb{P}(j \in \tau(T_n) \,|\, a_j(T_n) = 1,\ H,\ \mathcal{F}(n)) = \\ &= \begin{cases} p(T_n - H) & \text{if } H \leq T_n - T_j \\ 0 & \text{if } H > T_n - T_j \end{cases} . \end{aligned} \tag{2.4}$$

We will write $\mathbb{1}_A$ to denote the indicator random variable for event A. Then undoing the conditioning on H gives

$$\begin{aligned} &\mathbb{P}(j \in \tau(T_n) \,|\, a_j(T_n) = 1,\ \mathcal{F}(n)) = \\ &= \mathbb{E}_H \left[\mathbb{1}_{\{H \leq T_n - T_j\}}\, p(T_n - H) \right] \end{aligned} \tag{2.5}$$

where the expected value is taken over the distribution of H. We also have

$$\mathbb{P}(a_j(T_n) = 1 \,|\, \mathcal{F}(n)) = \mathbb{E}[a_j(T_n) \,|\, \mathcal{F}(n)] \tag{2.6}$$

and therefore

$$\begin{aligned} &\mathbb{P}(j \in \tau(T_n) \cap \mathcal{L}(T_n) \,|\, \mathcal{F}(n)) = \\ &= \mathbb{E}[a_j(T_n) \,|\, \mathcal{F}(n)] \; \mathbb{E}_H \left[\mathbb{1}_{\{H \le T_n - T_j\}} \, p(T_n - H) \right]. \end{aligned} \tag{2.7}$$

We now undo the conditioning on $\mathcal{F}(n)$ (the history of the DAG) and obtain

$$\begin{aligned} &\mathbb{P}(j \in \tau(T_n) \cap \mathcal{L}(T_n)) = \\ &= \mathbb{E} \left[a_j(T_n) \; \mathbb{1}_{\{H \le T_n - T_j\}} \, p(T_n - H) \right]. \end{aligned} \tag{2.8}$$

Note also that $a_j(T_{n+1}) - a_j(T_n) = -1$ if and only if the block j is a tip at time T_n and is approved by the new transaction which arrives at time T_n, and is zero otherwise. Therefore, we get

$$\begin{aligned} &\mathbb{E}[a_j(T_{n+1}) - a_j(T_n)] = \\ &= -\mathbb{E} \left[a_j(T_n) \; \mathbb{1}_{\{H \le T_n - T_j\}} \, p(T_n - H) \right]. \end{aligned} \tag{2.9}$$

2.2.2 The Fluid Limit

The fluid limit is reached as the arrival rate of new blocks, λ, goes to infinity. For convenience we will assume that the time between arrivals is fixed and equal to λ^{-1}, so $T_n = n\lambda^{-1}$, and we extend a_j to be piecewise constant in each interval $[T_n, T_{n+1})$. Given $s > 0$ let $m = \lfloor \lambda s \rfloor$, and define the set $\mathcal{A}(s) = \{m, m+1, \ldots, m+q\}$ where q is an integer depending on λ such that $\{q \to \infty$, $q\lambda^{-1} \to 0\}$ as $\lambda \to \infty$. We define

$$b(t,s) = q^{-1} \sum_{j \in \mathcal{A}(s)} a_j(t), \qquad l(t) = \lambda^{-1} \, L(t) \tag{2.10}$$

(assuming that $t \ge s + q\lambda^{-1}$ in b). Our main assumption for the fluid limit is that $b(t,s)$ and $l(t)$ converge to non-random differentiable functions as $\lambda \to \infty$. So, in particular we assume that

$$q^{-1} \sum_{j \in \mathcal{A}(s)} a_j(t) \to b(t,s) \quad \text{as } \lambda \to \infty. \tag{2.11}$$

We also assume that for all $j \in \mathcal{A}(s)$ and all $n \in \mathcal{A}(t)$, as $\lambda \to \infty$,

$$\begin{aligned} &q^{-1} \sum_{j \in \mathcal{A}(s)} a_j(T_n) \, \mathbb{1}_{\{H \le T_n - T_j\}} \, p(T_n - H) \\ &= \lambda^{-1} \, b(t,s) \, \mathbb{1}_{\{H \le t-s\}} \, \frac{2}{l(t-H)} + o(\lambda^{-1}). \end{aligned} \tag{2.12}$$

We now sum over $j \in \mathcal{A}(s)$ and $n \in \mathcal{A}(t)$ in (2.9), leading to

$$\begin{aligned} &\textstyle\sum_{n \in \mathcal{A}(t)} \, q^{-1} \, \sum_{j \in \mathcal{A}(s)} \mathbb{E}[a_j(T_{n+1}) - a_j(T_n)] = \\ &= \textstyle\sum_{n \in \mathcal{A}(t)} \, b(T_{n+1}, s) - b(T_n, s) = \\ &= b(t + q\lambda^{-1}, s) - b(t,s) = \\ &= - \textstyle\sum_{n \in \mathcal{A}(t)} \{ \lambda^{-1} \, b(t,s) \; \mathbb{E}_H \left[\mathbb{1}_{\{H \le t-s\}} \frac{2}{l(t-H)} \right] + \\ &+ o(\lambda^{-1}) \} = -q \, \lambda^{-1} \, b(t,s) \; \mathbb{E}_H \left[\mathbb{1}_{\{H \le t-s\}} \, \frac{2}{l(t-H)} \right] + \\ &+ o(q\lambda^{-1}). \end{aligned} \tag{2.13}$$

Since $q\lambda^{-1} \to 0$ as $\lambda \to \infty$, we get

$$\frac{\partial b}{\partial t}(t,s) = -b(t,s) \; \mathbb{E}_H \left[\mathbb{1}_{\{H \le t-s\}} \, \frac{2}{l(t-H)} \right]. \tag{2.14}$$

It is convenient to change variables at this point and define the age of the current tips to be $v = t - s$, and define a new density $g(t, v) = b(t, s)$, where $g(t, v)$ represents the density of tips present at time t which were added at time $t - v$ (or, equivalently, the density of tips present at time t with age v). Note also the total number of tips (rescaled by λ^{-1}) is

$$l(t) = \int_0^t b(t, s)\, ds = \int_0^t g(t, v)\, dv. \tag{2.15}$$

Furthermore the condition $a_n(T_n) = 1$ leads to the condition $b(t, t) = 1$, which in turn gives

$$g(t, 0) = 1. \tag{2.16}$$

In terms of these new variables, the fluid limit is described by the following set of equations:

$$\begin{aligned} \frac{\partial g}{\partial t} + \frac{\partial g}{\partial v} &= -g(t, v)\, \mathbb{E}_H \left[\mathbb{1}\{H \le v\} \left(\frac{2}{l(t - H)} \right) \right] \\ l(t) &= \int_0^t g(t, v)\, dv \\ g(t, 0) &= 1. \end{aligned} \tag{2.17}$$

The right side of (2.17) can be written more explicitly using the pdf for H. That is, assume that H is continuous with pdf $f(x)$, then (2.17) can be written

$$\frac{\partial g}{\partial t} + \frac{\partial g}{\partial v} = -g(t, v) \int_0^v \frac{2}{l(t - x)}\, f(x)\, dx. \tag{2.18}$$

2.2.3 Comparison with Previous Work for Fixed Delay

In previous work [1], the case of fixed delay $H = h$ was analysed by different means, and it was shown that the fluid limit was described by a delay differential equation. Here we compare that result with the present work. For the case of constant $H = h$, the PDE (2.17) leads to the equation

$$\frac{dl}{dt} = \begin{cases} 1 & t < h \\ 1 - \dfrac{2}{l(t - h)} \displaystyle\int_h^t g(t, v)\, dv & t \ge h \end{cases}. \tag{2.19}$$

In [1] the following equation was derived:

$$\frac{dl}{dt} = 1 - \frac{2}{l(t - h)}\, x(t - h) \tag{2.20}$$

where $x(s)$ is the number of 'free' tips at time s, which is the number of tips that have not yet been selected at time s for validation by any newly created blocks. Since the validation time is fixed to be h, it follows that all these free tips at time $t - h$ must still be tips at time t. Furthermore any 'pending' tips at time $t - h$ will no longer be tips at time t. Thus the set of tips at time t will consist of the free tips at time $t - h$, plus any additional tips that arrived in the time interval $[t - h, t]$. These latter tips are the tips whose age is less than h, thus we can write

$$l(t) = x(t - h) + \int_0^h g(t, v)\, dv. \tag{2.21}$$

Combining with the relation $l(t) = \int_0^t g(t, v)\, dv$ we deduce that for $t \ge h$

$$x(t - h) = \int_h^t g(t, v)\, dv. \tag{2.22}$$

Therefore, the two expressions (2.19) and (2.20) are identical for all $t \ge h$. Hence the method presented here leads to the same result as the method from [1].

2.3 The Stationary Solution

We expect that the solution of the system (2.17) will converge to a time-independent solution as $t \to \infty$. We can compute this time-independent solution: assume that there is a function $g(v)$ and constant l such that

$$g(t, v) \to g(v), \quad l(t) \to l \quad \text{as } t \to \infty. \tag{2.23}$$

Substituting in (2.17) we find

$$g'(v) = -g(v)\,\frac{2}{l}\,\mathbb{P}(H \leq v), \quad g(0) = 1 \tag{2.24}$$

which leads to

$$g(v) = \exp\left[-\frac{2}{l}\int_0^v \mathbb{P}(H \leq u)\,du\right]. \tag{2.25}$$

This can be used to get an implicit equation for l:

$$l = \int_0^\infty g(v)\,dv. \tag{2.26}$$

2.3.1 Simulator Description

In what follows, we consider some specific examples for delay distributions. For each of these special cases, we derive the corresponding tip equilibrium and then verify this result with Monte Carlo simulations[7]. We simulate a single node and we generate new blocks according to a Poisson arrival process with rate λ, where λ is specified for each experiment. Blocks are produced by a large number of independent agents which makes the Poisson process a natural and standard choice for generating arrivals. The simulations operate in discrete time increments of 0.1 seconds. When a new block is generated, tips are selected from the node's current tip set, but the new block does not become visible to the node until the delay time h for the new block has passed. The delay, h, for each block is randomly drawn from the chosen distribution of H in each experiment.

2.3.2 Special Case: Fixed Delay $H = h$

Here we assume that $H = h$ is constant for all tips. Then solving (2.24) we get

$$g(v) = \begin{cases} 1 & v \leq h \\ e^{-2(v-h)/l} & v > h \end{cases}. \tag{2.27}$$

We can also use (2.26) to compute l: this gives

$$l = \int_0^\infty g(v)\,dv = h + \int_0^\infty e^{-2(v-h)/l}\,dv = h + \frac{l}{2} \tag{2.28}$$

which leads to the value

$$l = 2h. \tag{2.29}$$

Figure 2.5 shows 150 Monte Carlo simulations of the DAG with fixed delay ($\lambda = 20, h = 5$). Note that the average value corresponds to $L = \lambda l = 2\lambda h = 200$.

2.3.3 Special Case: Exponential Delay H

Here we assume that H is exponential with rate μ, so that (2.24) is

$$g'(v) = -g(v)\,\frac{2}{l}\,(1 - e^{-\mu v}), \quad g(0) = 1. \tag{2.30}$$

[7] Source code available at `https://github.com/cyberphysic4l/iota-sim`.

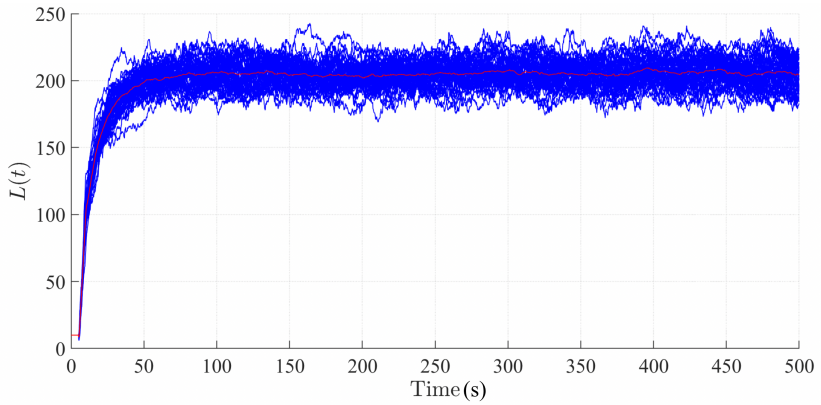


Figure 2.5: 150 Monte Carlo simulations of the DAG with constant delay ($\lambda = 20, h = 5$). The single realisations are shown in blue, while the average value is shown in red. Notice that we obtain the predicted average value $L = 200$.

This leads to the solution

$$g(v) = \exp\left[\frac{2}{l}\left(v + \mu^{-1}e^{-\mu v} - \mu^{-1}\right)\right]. \tag{2.31}$$

Using (2.26) we can compute l. With $h = \mu^{-1}$ this gives

$$l = 1.2839\, h. \tag{2.32}$$

Figure 2.6 shows 150 Monte Carlo simulations of a DAG with exponential delay ($\lambda = 20, \mu = 0.2 = 5^{-1} = h^{-1}$). Note that the average value corresponds to $L = \lambda l = 1.28\, \lambda\, h = 128$.

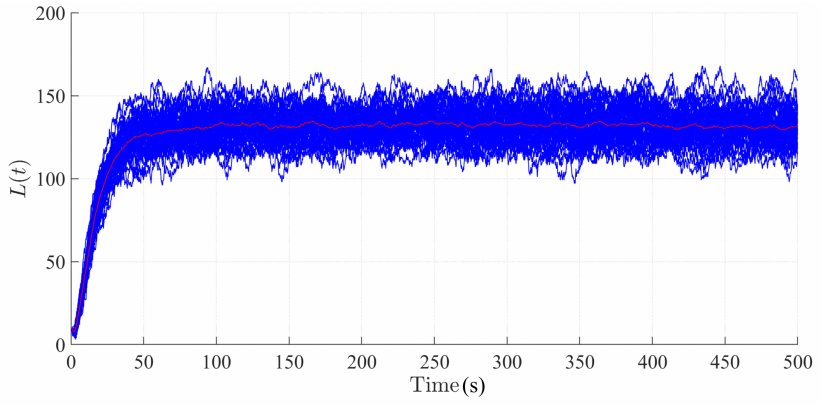


Figure 2.6: 150 Monte Carlo simulations of a DAG-based ledger with exponential delay ($\lambda = 20, \mu = 0.2 = 5^{-1} = h^{-1}$). The single realisations are shown in blue, while the average value is shown in red. Notice that we obtain the predicted average $L = 128$.

2.3.4 Special Case: Uniform Delay H

Here we assume that H is uniform on some interval $[h_0, h_1]$. This gives

$$\int_0^v \mathbb{P}(H \leq u)\, du = \begin{cases} 0 & v \leq h_0 \\ \dfrac{(v-h_0)^2}{2(h_1-h_0)} & h_0 \leq v \leq h_1 \\ v - \dfrac{h_0+h_1}{2} & v > h_1 \end{cases}. \tag{2.33}$$

Also we define $\beta = \sqrt{h_1 - h_0}$ then the equation for l is

$$l = h_0 + \frac{l}{2}\, e^{-\beta^2/l} + \beta \int_0^\beta e^{-w^2/l}\, dw. \tag{2.34}$$

One particular case: $h_0 = 1$, $h_1 = 11$, gives $l = 10.69$. In terms of the mean delay $h = 6$ this is

$$l = 1.782\, h. \tag{2.35}$$

Figure 2.7 shows 150 Monte Carlo simulations of the DAG with uniform delay ($\lambda = 20, h_0 = 1$, $h_1 = 11$). Note that the average value corresponds to $L = \lambda l = 1.78\, \lambda h = 214$.

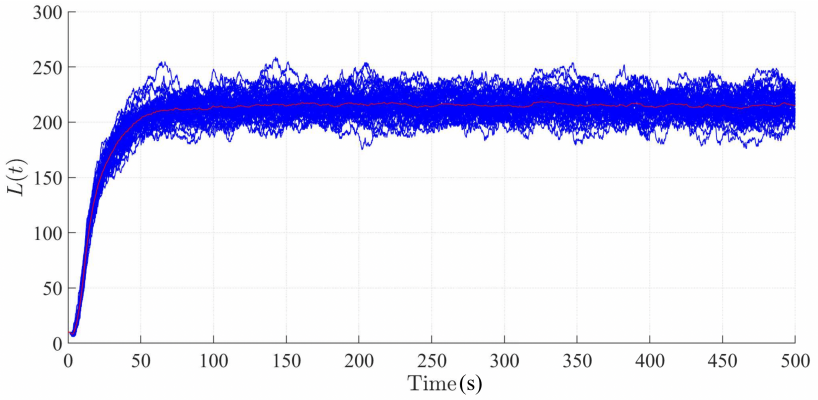


Figure 2.7: 150 Monte Carlo simulations of a DAG-based ledger with uniform delay ($\lambda = 20, h_0 = 1$, $h_1 = 11$). The single realisations are shown in blue, while the average value is shown in red. Notice that we obtain the predicted average $L = 214$.

2.4 Chapter Summary

In this chapter, we have introduced DAG-based distributed ledgers which represent a natural extension of the blockchain as we have highlighted with reference to their core constituents. We presented a fluid model for the evolution of a DAG under URTS and variable network delays. We then verified our model's accuracy for a number of special cases of delay distributions. At the time of this work's original publication [53], PoW was a necessary feature of any DAG-based DLT and it was generally expected that the primary source of delays in DAG-based ledgers could be attributed to the process of finding a PoW solution. The exponential delay model presented here can be expected to accurately model this case. However, since [53], new approaches to access control have been developed, removing the need for PoW [8]. The time-consuming processes which determine delay distributions in these new DLT networks are more complex and distributed than those of PoW ledgers, but the modelling work presented in this chapter has contributed greatly to the analysis of measurements and prediction of new behaviour in these more complex networks.

Future work on variable delay models should be focused on finding appropriate delay distributions to model real modern DLT networks and to take into account modifications of the basic URTS algorithm for tip selection.

Chapter 3

Parasite Chain Attacks and Tip Selection Algorithm Design

***Abstract*— Secure distributed ledgers should operate as immutable records and it should be impossible for any malicious agent to reverse or erase any data that has been confirmed by the network. It is customary in the domain of cyber security to test new architectures by studying their resilience against specific attacks which attempt to break the network. One such attack on DAG-based distributed ledgers, known as a *parasite chain attack*, seeks to reverse blocks that have been confirmed by the network, allowing the attacker to spend the same token twice. The general class of attacks to which parasite chains belong, known as double spending attacks, are straightforward to analyse for blockchains but are more complex and varied in the case of DAGs. In this chapter, we present a model for BRW tip selection on DAG-based ledgers and analyse parasite chain attacks. Our results provide useful insights to guide the design of tip selection algorithms with a view to preventing such attacks. Finally, we present a new variant of BRW tip selection inspired by the results of our analysis which we show to offer improved security against these attacks. Part of the work presented in this chapter also appears in [55].**

Distibuted ledgers are record-keeping tools which allow untrusting agents to interact with one another without the need for a trusted intermediary. A simple example of how a distributed ledger can be used is to transfer funds from a customer to a merchant in exchange for some goods. To achieve this, the customer adds a block to the ledger containing a transaction which transfers funds to the merchant's account. The merchant waits for the block to be confirmed and then hands over the goods to the customer, with the understanding that the funds are now irreversibly recorded on the ledger to be in the merchant's account. If the customer could then somehow reverse this transaction by creating a conflicting version of the ledger which the network then confirms instead, we would refer to this as a double spend. This essentially equates to the customer stealing from the merchant, so in order for a distributed ledger to be deemed secure it must be resilient against double spend attacks.

Let us consider an example of double spending in DAG-based DLTs to develop the idea further: Figure 3.1 shows an instance of a DAG-based distributed ledger. A malicious node adds some data to the ledger which we represent as the yellow block in the figure. The same node subsequently writes multiple blocks to the ledger, represented by green blocks, which contain data conflicting with the yellow block. It is worth stressing, at this point, that there is no mechanism to force a user to select certain blocks for approval. Any pair of blocks can be selected as long as they are mutually consistent with all blocks they approve (directly or indirectly), in other words, all blocks in the past cone of the new blocks must be without conflicting data. In this scenario, all the blocks that approve the original yellow block (the blue blocks) are inconsistent with the green ones, and therefore any new blocks can either approve the green/black blocks or the blue/black

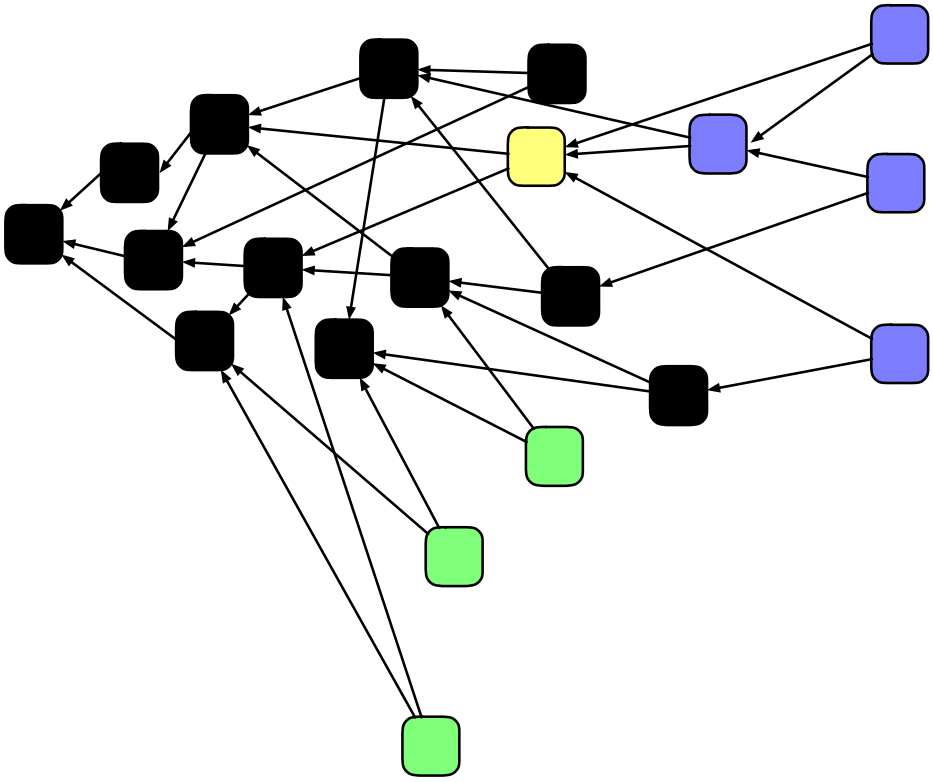


Figure 3.1: The blue and the green transactions are incompatible with each other. This image was also present in [1].

ones. The green/blue combination would be considered invalid (as there is an inconsistency in the ledger) and a new selection would be made. The objective of a double spend attacker would be to publish the yellow block and for this to be acted on in some way, for example, if the yellow block transfers token to a merchant, the attacker would wait for this transaction to be confirmed and goods to be transferred by the merchant. The attacker would then release the green blocks to the network which spend the same tokens as the yellow block in such a way that they get approved by the majority of the network rather than the original data (thereby reversing the record of the data that has already been acted upon). However, it is not straightforward for an attacker to convince the majority of other nodes to approve the double spend blocks: it requires careful choice of the structure of the conflicting sub-DAG, timing of publishing the conflicting blocks, and requires consumption of a great deal of resources. The parasite chain attack which we discuss in this chapter exemplifies an effective and efficient approach that an attacker could take to maximise its chances of succeeding.

The contributions of this chapter can be briefly summarised as below.

- A mathematical description of the DAG, together with a Markov chain model for BRW[8] tip selection algorithms. This model allows for rapid computation of the expected outcome of random walks and hence the probability of the success of an attack on a given DAG. The model may also be used to efficiently implement the BRW algorithm in practice.

- An analysis of the *parasite chain attack*, wherein the attacker attempts to alter or reverse one or more transactions previously added to the ledger. The insights gained from this analysis can be used to guide parameter selection when designing a real DAG-based DLT network to

[8] see Section 2.1.2 for an introductory discussion of BRW tip selection

be resilient against attacks of this kind.

- A proposal for a new variant of the BRW tip selection algorithm, which is shown to improve the resistance of the ledger to parasite chain attacks. This improved tip selection algorithm is inspired by the results of our analysis and simulations.

The remainder of this chapter is organised as follows. In Section 3.1 we introduce a double spending mechanism known as a parasite chain attack. Section 3.2 summarises the stochastic model for the DAG and presents a new formulation for the BRW tip selection algorithm as an absorbing Markov chain. This formulation is used to analyse the algorithm's resistance to parasite chain attacks over a range of parameter choices and can also be used as an efficient implementation of the BRW algorithm in practice. Section 3.4 introduces a modification to the BRW which makes use of the growth of the cumulative weight in the DAG and presents results showing its improved resilience against parasite chain attacks.

3.1 The Parasite Chain Attack

To see how in practice a hypothetical attacker could carry out a double spending attack in a DAG-based ledger, we consider the attack scenario known as a *parasite chain attack*. A simple[9] parasite chain (SPC) is illustrated in Figure 3.2: we refer to this as k^{th}-order SPC because the first k transactions in the chain reference the main DAG . The attacker publishes the yellow block in the DAG and simultaneously, in secret, creates a conflicting block (the green one) followed by a chain of blocks which validate it. The attacker waits for the yellow block to be confirmed[10] and then broadcasts the parasite chain to its neighbouring nodes and continues publishing blocks which validate it. Recall from the brief description in Section 2.1.2 that the BRW tip selection algorithm employed by honest nodes favours heavier branches of the DAG, so the goal of the attacker is then to "race" the main DAG, creating a sub-DAG (parasite chain) whose cumulative weight will outmatch the main body of the ledger. If the attack were to succeed, the parasite chain would become the main DAG which the majority of nodes attach their blocks to and the original yellow block would be reverted along with the sub-DAG that approves it.

An SPC can be characterised by three parameters:

1) T_{DS} is the time between the arrival of the original block and the broadcast of the SPC;

2) k is the 'order' of the SPC, i.e. the number of blocks in the chain referencing the main DAG;

3) μ is the rate at which the attacker can add blocks to the parasite chain.

It is important to remark that T_{DS} and k can be freely chosen by the attacker, but μ is determined by the resources available to them which are difficult or expensive to obtain. For example, in a PoW-based network, μ is determined by the computing power available to the malicious node, whilst in a PoR-based network, μ is determined by the attacker's reputation (e.g., wealth).

Note that it is possible to create parasite chains with more complex structure than the one presented above. However due to the complexity of the analysis it is not yet clear how to optimally design such a structure. Therefore, in the remainder of this chapter we will focus solely on the SPC, and use this special case to draw conclusions about the DAG's ability to resist double spending attacks.

[9] We refer to the SPC as simple because we assume that the chain is attached at a single point on the main DAG in order to simplify analysis.

[10] Recall that, in general, a block must receive a minimum number of approvals before it is considered to have been *confirmed*. This ensures that some majority of nodes agree with this block and that it is very unlikely to be reverted. A merchant will not accept a transaction until it is confirmed, so the attacker should wait at least until the first block is confirmed before broadcasting the SPC and attempting to revert it.

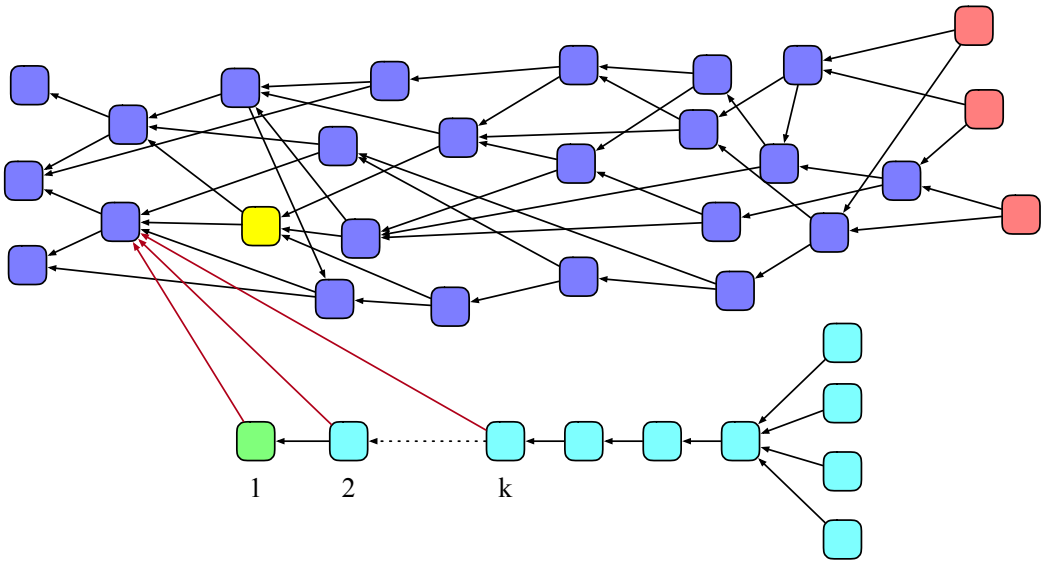


Figure 3.2: A k^{th}-order simple parasite chain: The yellow and green blocks constitute a double spend.

3.2 The Biased Random Walk Algorithm

In this section, we present a Markov chain model for the BRW algorithm which allows us to compute the probability that a BRW will terminate on a given tip of a DAG. We consider a single local copy of the ledger, and in order to generate a random instance thereof, we use an agent based model: at each time step a random number of blocks are generated, according to a Poisson distribution with rate λ; an agent selects two random blocks from the DAG via the BRW tip selection algorithm and references these in the new block; the block appears in the ledger and becomes available for subsequent tip selection after a short time delay which simulates processes such as PoW and dissemination of the block through the network.

3.2.1 Matrix Model

First we recall the description of the agent-based model which is used to generate a random instance of the DAG. Each new transaction selects two tips for approval, and attempts to validate them. To take into account possible conflicts we assume that d conflicting sub-DAGs exist on the DAG, where each block from a sub-DAG is mutually consistent only with transactions from the same sub-DAG. Thus every block has a label from the set $1, \ldots, d$, indicating the sub-DAG to which it belongs. We will call this the *type* of the block. If validation fails (i.e., transactions from different sub-DAGs are selected) both choices are discarded and another two tips are selected for validation. This continues until the process is successful, and we assume that this whole validation effort is essentially instantaneous. After the validation there is a delay h before the new block becomes visible to the network. During this time the approvals of the selected tips are pending, so the tips may still be available for selection by other new transactions. After the waiting time h the two blocks which were successfully approved are removed from the tips set, and so are no longer available for selection by other new blocks (at least, by the ones that follow the protocol)[11].

Next we review the BRW algorithm for tip selection, initially described in Section 2.1.2: for a given DAG instance, a random walk is initiated starting somewhere in the interior of the graph, and the BRW subsequently jumps randomly along its edges. A jump along an edge can be either forward (meaning from an older block to a newer block) or backward. A forward jump from a block i to a block j occurs with a probability that is proportional to $\exp(-\alpha(\mathcal{H}_i - \mathcal{H}_j))$, where α is a positive tuning parameter of the BRW algorithm. The walk terminates when it reaches a tip,

[11] It may happen that some of these blocks had already ceased to be tips at an earlier time, due to their being validated by some other new block.

which is then selected for approval. Thus, to model the BRW algorithm we need to define explicitly the cumulative weight of each block and find an expression for the probability of terminating on a given tip.

Let $B(t)$ denote the set of blocks in the DAG at time t. The cumulative weight $\mathcal{H}_i(t)$ of block i is defined as the number of blocks that directly or indirectly approve it at time t:

$$\mathcal{H}_i(t) = \#\{z \in B(t) : z \text{ approves } i\}. \tag{3.1}$$

The weight $\mathcal{H}_i(t)$ can be computed using the adjacency matrix $M(t)$ of the DAG, by noting that $[M^k(t)]_{ij}$ represents the number of paths of length k that connect block i to block j. In what follows, we assume that block indexes provide a total ordering such that if block i arrived earlier than block j, then $i < j$. Let t_i denote the time at which block i appears in the DAG and becomes available for tip selection. Furthermore, define

$$\kappa_i(t) = \lfloor (t - t_i)/h \rfloor, \tag{3.2}$$

$$P_i(t) = \sum_{k=1}^{\kappa_i(t)} M^k(t), \tag{3.3}$$

where $\lfloor \cdot \rfloor$ is the floor operator. Then $\mathcal{H}_i(t)$ is equal to

$$\mathcal{H}_i(t) = 1 + \sum_{j=i+1}^{N(t)} \min\{\mathbf{e_i}^T P_i(t) \mathbf{e_j}, 1\} \tag{3.4}$$

where $\mathbf{e_i}$ is the i-th vector of the canonical orthonormal basis. Notice that:

- we use $\min\{\cdot, 1\}$ in order to avoid counting the same block more than once (since there could be several paths with a different number of steps connecting two blocks);
- the value $\kappa_i(t)$ represents the maximum number of forward jumps that can occur along a directed path from block i to the tips set (as each new block take h seconds to become visible and available for tip selection).

Given (3.4), we can go back to the BRW algorithm: in the most general case, the random walk starts at a random block and we can define $\pi \in \mathbb{R}^{N(t)}$ as the vector whose i-th entry represents the probability that the walk starts at block i. Furthermore we define the difference in cumulative weight between block j and block k to be $\vartheta_{jk}(t) = \mathcal{H}_j(t) - \mathcal{H}_k(t)$. Then the transition matrix $T(t)$ whose jk entry characterises the probability of jumping from block j to block k is defined as follows in the case where j is not a tip:

$$[T]_{jk}(t) = \begin{cases} q/m & \text{if } k \in \mathcal{P}_j \\ (1-q)\dfrac{e^{-\alpha\vartheta_{jk}(t)}}{\sum_{z \in I_j} e^{-\alpha\vartheta_{jz}(t)}} & \text{if } k \in C_j \\ 0 & \text{otherwise} \end{cases} \tag{3.5}$$

where $q \in [0, 1/2)$ represents the probability of going backwards, $\mathcal{P}_j \subset B(t)$ is the set of all parents of block j, $m = |\mathcal{P}_j|$ is the number of parents of block j, $C_j \subset B(t)$ is the set of all children of block j, and α is a positive tuning parameter. When the walk hits a tip, it remains there indefinitely, therefore when the transaction j is a tip we have

$$[T]_{jk}(t) = \begin{cases} 1 & \text{if } j = k \\ 0 & \text{otherwise} \end{cases}. \tag{3.6}$$

(3.5) and (3.6) provide us with useful information on the BRW algorithm: the jumping process is an absorbing Markov chain with $N(t) - L(t)$ transient states and $L(t)$ absorbing states, where $N(t) = |B(t)|$ is the number of blocks in the DAG at time t, and $L(t)$ is the number of tips at time t. Therefore, by properly rearranging and re-labelling the blocks, the transition matrix can be written as follows:

$$T(t) = \left(\begin{array}{c|c} Q(t) & R(t) \\ \hline 0 & I \end{array}\right)$$

where $Q(t)$ is the transition matrix between transient states, $R(t)$ is the transition matrix from transient states to absorbing states, and I is the identity matrix for the absorbing states. Standard analysis of absorbing Markov chains yields the absorbing probability matrix $B(t) = (I - Q(t))^{-1}R(t)$, whose (i, j) entry is the probability for the BRW to be absorbed at tip j given that it started at block i.

Next we include the effects of conflicting sub-DAGs. Define $\mathcal{L}_i(t)$ to be the set containing the indices of the tips of type i at time t, and recall that π is the initial probability distribution of the random walk. Then the probability for the random walk to terminate at a tip of type i (i.e., to be absorbed by the subset $\mathcal{L}_i$ of the absorbing states) is

$$p_i(t) = \frac{\sum_{j\in\mathcal{L}_i(t)} \pi^T B\mathbf{e_j}}{\sum_{k=1}^{d}[\sum_{j\in\mathcal{L}_k(t)} \pi^T B\mathbf{e_j}]}. \tag{3.7}$$

Furthermore the probability for a new block at time t to join the tip set of type i is then

$$P(\text{Type} = \text{i}) = p_i(t)^m \left(\sum_{j=1}^{d} p_j(t)^m\right)^{-1} \tag{3.8}$$

where the second factor on the right side of (3.8) accounts for the requirement that all m selections must have the same type.

The validity of (3.7) was investigated using a Monte Carlo analysis of random walks on a randomly generated DAG with two sub-DAGs. In each run of the Monte Carlo analysis a random walk was generated starting at the genesis, and the type of the terminal tip of the walk was recorded as either Type 1 or Type 2. We observe that as the number of Monte Carlo simulations increases the empirical probability distribution of the BRW's type converges to the value computed by (3.7). This is illustrated in Figure 3.3. Two sub-DAGs of similar size and structure were generated for the simulation shown in Figure 3.3, so the probability of the BRW terminating on either type is close to 0.5 (p_2, the probability of terminating on a tip of the second sub-DAG, is shown here).

3.3 Resistance to Parasite Chain Attacks

In this section, we report on our investigations of the SPC (see Figure 3.2) attack on a DAG-based ledger, when the BRW algorithm is used to select tips[12]. In particular we are interested in how the parameters of the SPC and of the BRW algorithm affect the likelihood of a BRW terminating on a dishonest tip. Notice that for an SPC attack to succeed, the majority of newly arriving blocks must eventually validate tips on the SPC. While it should be worthwhile to investigate the probability of this event, it would require an analysis of the whole dynamical system and of its equilibrium states, and would stretch beyond the scope of this work. Furthermore, the probability of this event would be closely related to the probability of a single BRW selecting a tip of the SPC, which is the focus of the analysis presented here.

We denote tips on the main DAG (which reference the first of the double spend transactions) as Type 1, and tips on the SPC as Type 2. Then (3.8) gives us:

[12]Source code available at `https://github.com/cyberphysic4l/iota-sim`.

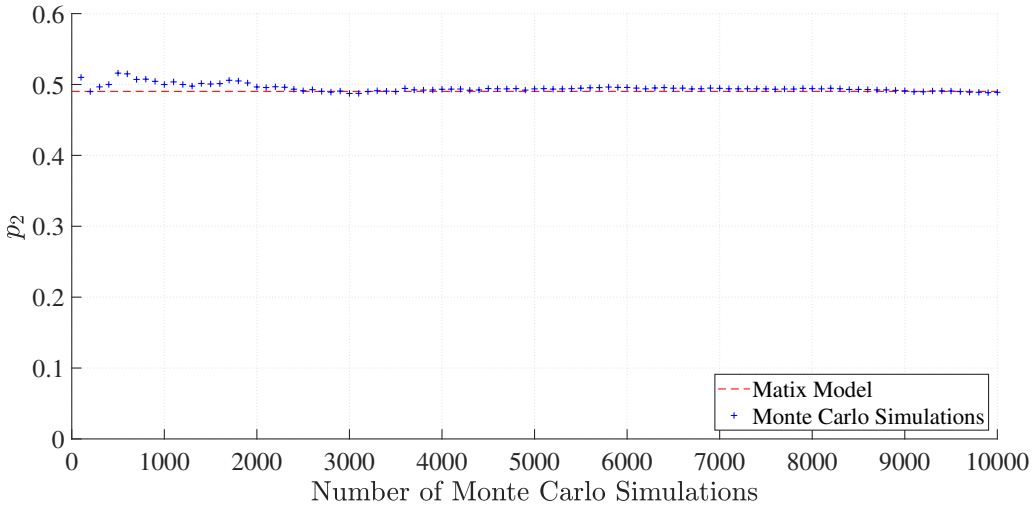


Figure 3.3: Convergence of BRW Monte Carlo simulation results to matrix model formula.

$$p_2(t) = \frac{\sum_{j\in\mathcal{L}_2(t)} \pi^T B\mathbf{e_j}}{\sum_{l\in\mathcal{L}_1(t)} \pi^T B\mathbf{e_l} + \sum_{j\in\mathcal{L}_2(t)} \pi^T B\mathbf{e_j}}. \tag{3.9}$$

We first investigate the effect of the parameter α by simulating this probability for a randomly generated instance of a DAG and a first order SPC with $T_{DS} = 120$ seconds and where the attacker has 25% of the total network access resources (i.e., $\mu = \lambda/3$). Results are shown in Figure 3.4. It can be seen from this figure that as α increases, the probability of selecting the SPC increases at first (from a nonzero value), reaches a maximum, and then decreases again: the reason for this behaviour is that as α approaches infinity the BRW will always move in the direction of the greatest increase in cumulative weight and hence always choose tips on the heaviest branch of the DAG. Therefore, as long as $\mu < \lambda$ and nodes wait long enough to confirm blocks, the attack will always fail. Note that since the attacker's aim is to be able to effectively spend its money twice, the attack would fail if they revealed the parasite chain before the original block had gained sufficient cumulative weight to be confirmed by other nodes. This would happen because if the original transaction was orphaned before its confirmation, then it would not be considered valid by the network and the second spend transaction would become the only transaction in which the attacker was able to successfully spend any currency. Accordingly, in the simulations presented in Figure 3.4, we assume that by $T_{DS} = 120$ the initial spend on the main branch will be confirmed and, consequently, acted upon. On the other hand, when α is equal to zero the selection will not depend on the cumulative weight but only on the structure of the graph, in which case an SPC attack is also unlikely to succeed because the attack is only attached at a single point and is therefore less likely to be selected. However, slightly more complicated attacks with multiple attachment points could very easily succeed in this case.

Next, we investigated the effect of the double spend time, T_{DS}, with some results depicted in Figure 3.5. Recall that in the simulations presented in Figure 3.4, we took $T_{DS} = 120$, i.e. we assumed that the transaction will be confirmed and acted upon within this time. The reason for the evident decrease in probability of selecting an SPC tip as T_{DS} increases, as shown in Figure 3.5, is related to how the cumulative weights of blocks grow. A block in the main DAG must wait for an adaptation period before *all* new arriving blocks will indirectly approve it, so the cumulative weight initially grows slowly and then gradually increases to grow linearly with rate λ. However, due to the chain structure of the SPC, each SPC block references the second spend block from the outset, and hence its cumulative weight will immediately grow linearly at the rate of arrival of the attacker's blocks, namely μ, with no adaptation period. This phenomenon is illustrated in Figure 3.6 which plots the average cumulative weight trajectories of 100 blocks of each type in a simulated DAG with an SPC. Additionally, Figure 3.7 provides an approximate illustration of the

DAG and SPC structures that result in this cumulative weight growth profile. The intersection point of the two traces in Figure 3.6 at around 90 seconds indicates that if T_{DS} could be chosen to be less than this, the cumulative weight of the second spend would be larger than the first, and selecting the SPC would be highly likely. Indeed, Figure 3.8 confirms that for $T_{DS} = 60$ increasing α leads to an increase in the probability of selecting a tip that belongs to the double spending sub-DAG. However, provided a merchant waits sufficiently long to confirm the first spend, this apparent advantage to the attacker will be of no use as the attacker must wait at least T_{DS} seconds for the first spend to be confirmed before releasing the SPC.

The results presented above not only demonstrate how parameter choices of the BRW tip selection affect success rates of SPC attacks, but also provide a more general intuition about how an honest node should decide when to consider a block confirmed in a given DAG-based DLT network. In particular, recall that Figure 3.5 shows us that the probability of an attack succeeding decreases if the attacker must wait longer to reveal the parasite chain, and furthermore, Figure 3.8 teaches us that an attack can be highly likely to succeed if the attacker can release their SPC early. Therefore, it is crucial to ensure that blocks are not confirmed too soon as this could allow an attacker to subsequently reverse them with a double spend and an SPC. The reason a parasite chain is more likely to be selected when released early is that the chain structure allows all attacker blocks to immediately add to the cumulative weight, whilst blocks in the main DAG take some time to be referenced by all new blocks, as depicted in Figure 3.6. The growth of cumulative weight over time of any block in a DAG depends on the structure of that DAG, which in turn depends on a variety of factors associated to a given DLT network such as the network delay, the computations required for each block and the tip selection strategies. As such, this growth is difficult to predict, however, it can be readily measured in a real network and one could produce a plot comparable to Figure 3.6 for a predicted attacker power. This plot could then be used to select an appropriate confirmation criterion by observing the intersection of these two traces and ensuring the attacker is forced to release their SPC after this time. For example, the nodes in the experiments presented here could choose to consider a block confirmed when its cumulative weight exceeds 1000—we see in Figure 3.6 that blocks in the main DAG reach this cumulative weight after around 130 seconds, so the attacker would be forced to wait at least this long to release their parasite chain.

The other key parameter of the SPC is the order, k, which indicates the number of blocks in the SPC which reference the main DAG. It is worth noting that adding additional references to the main DAG does not cost the attacker any additional computational resources, so they do not need to consider this as a factor in their choice of k. There are, however, two key factors which will influence the attackers optimal choice of k. The larger the choice of k, the more likely the BRW is to jump on to the parasite chain from the main DAG at the attachment point. However, more links also means more opportunities to jump back on to main DAG from the SPC (since at every step the walk may backstep with probability q). The relationship between the probability of selecting an SPC tip, the attacker's choice of k, and the BRW backstepping probability parameter q is illustrated in Figure 3.9.

3.4 Extending the BRW Algorithm

The original motivation for using the BRW selection algorithm was to incentivise network users to validate the most recently arrived tips, and thereby defend the DAG against attacks such as the parasite chain attack. Theoretically, if the BRW parameter α is high enough and if the attacker does not possess the majority of the resources of the network, then the attack should never succeed. However an excessively high value of α would result in many honest blocks never being approved. The design trade-off between security and liveness involved in choosing α is discussed in [32], and solutions to the issue of blocks being orphaned are proposed in [31].

We now propose a modification to the BRW algorithm which seeks to reduce the efficacy of double spending attacks whilst allowing us to maintain a low α, and hence a *wide* DAG in which there is a low probability of blocks being orphaned. The intuition for our modification stems from the phenomenon illustrated in Figure 3.6: although the cumulative weight of a block on the main sub-DAG may lag behind due to the initial adaptation period it underwent, we can be quite sure that if the point of attachment of a parasite chain is deep in the DAG, then the rate of growth of

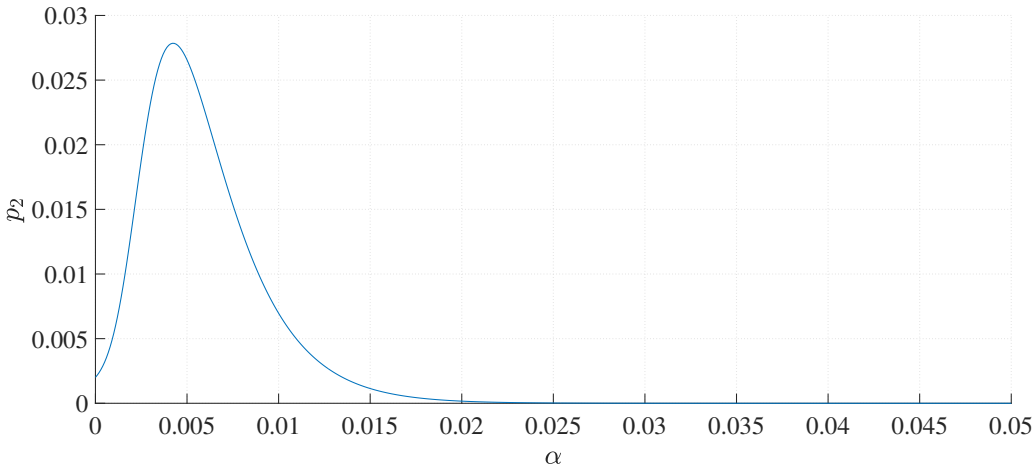


Figure 3.4: Probability of selecting an SPC tip: $\lambda = 15$, $\mu = 5$, $k = 1$, $T_{DS} = 120$.

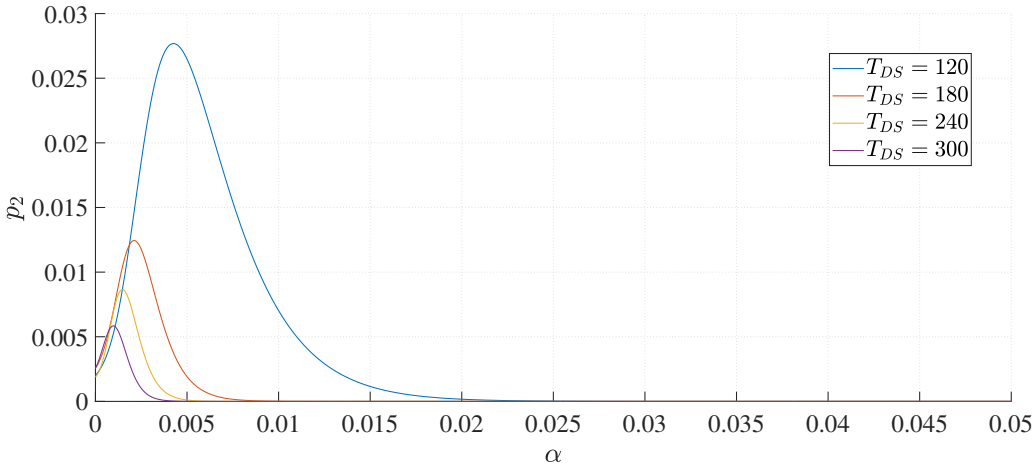


Figure 3.5: Probability of selecting an SPC tip: $\lambda = 15$, $\mu = 5$, $k = 1$.

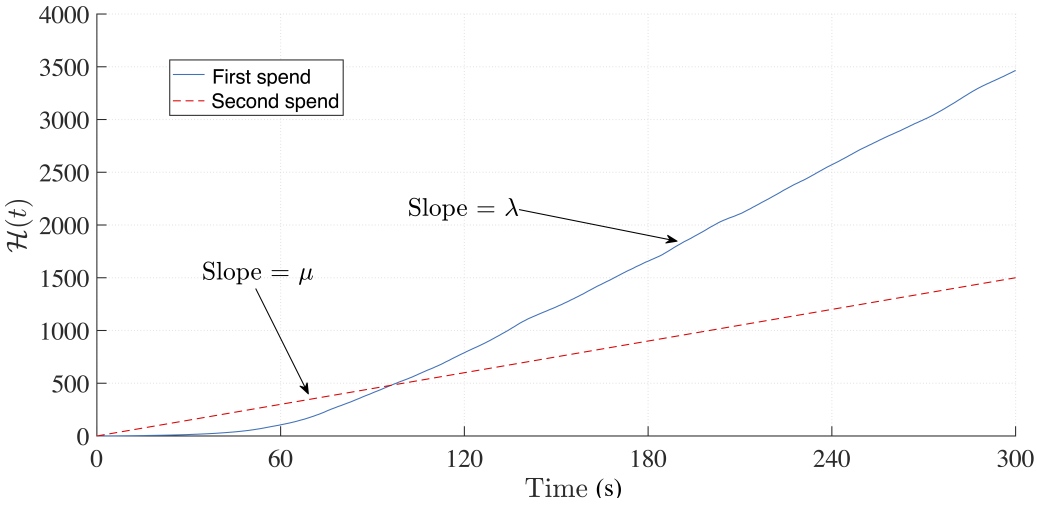


Figure 3.6: Growth of cumulative weight in the main sub-DAG and SPC.

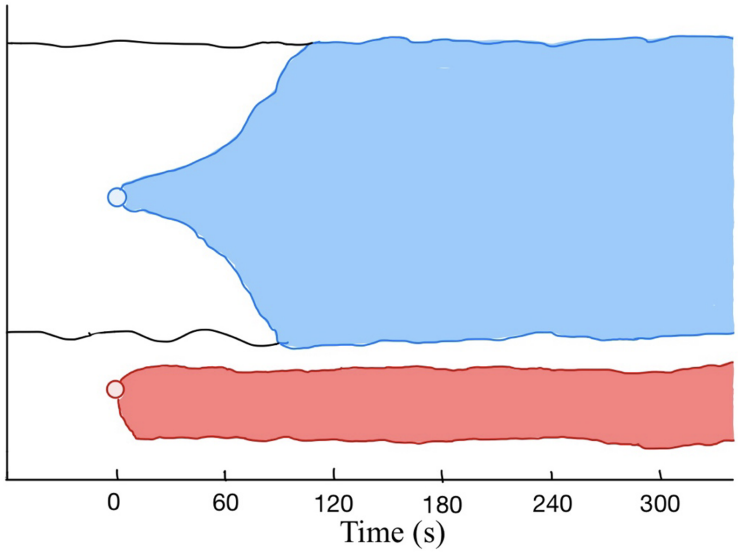


Figure 3.7: An approximate illustration of how cumulative weight grows in a DAG (blue) and a parasite chain (red). The blue and red circles at time zero represent the first and second spends, respectively, and the thickness of the shaded area represents the rate of new blocks that reference these spends as they are attached to each structure.

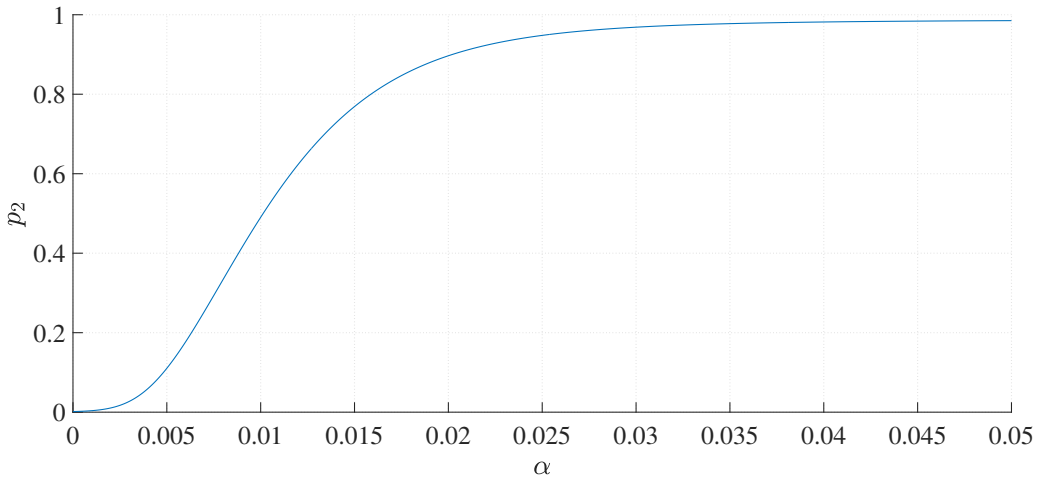


Figure 3.8: Probability of selecting an SPC tip: $\lambda = 15$, $\mu = 5$, $k = 1$, $T_{DS} = 60$.

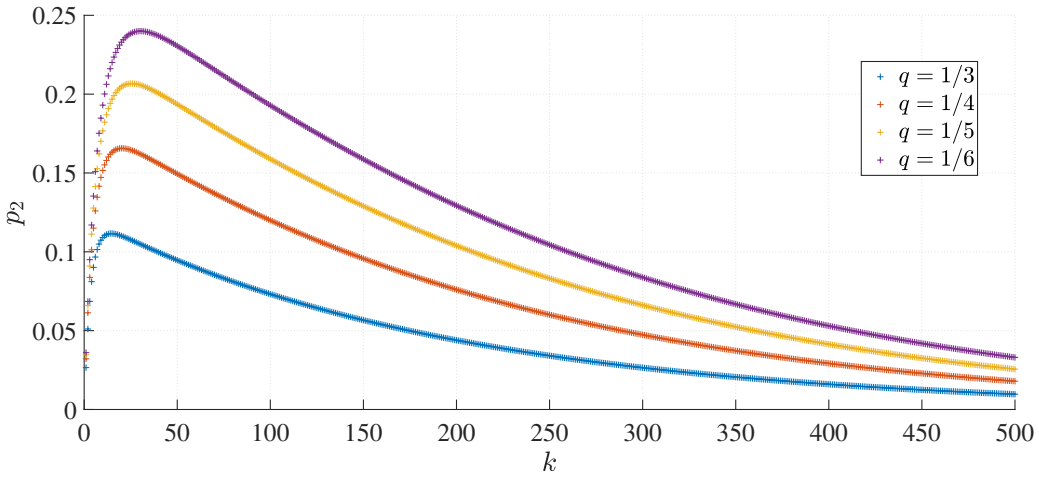


Figure 3.9: Probability of selecting an SPC tip: $\lambda = 15$, $\mu = 5$, $\alpha = 0.005$, $T_{DS} = 120$.

the cumulative weight of blocks at the attachment site should be equal to λ (the slope of the blue curve in Figure 3.6). Our modification utilises the first order time derivative of the cumulative weight in calculating the jumping probabilities of the BRW—we call this a *First Order* BRW. Define $\vartheta^{(1)}_{jk}(t) = |\mathcal{H}'_j(t) - \mathcal{H}'_k(t)|$, where $\mathcal{H}'_k(t)$ is the time derivative of the cumulative weight:

$$[T]_{jk}(t) = \begin{cases} q/m & \text{if } k \in \mathcal{P}_j \\ (1-q)\dfrac{e^{-\alpha\vartheta_{jk}(t)-\beta\vartheta^{(1)}_{jk}(t)}}{\sum_{z\in I_j} e^{-\alpha\vartheta_{jz}(t)-\beta\vartheta^{(1)}_{jz}(t)}} & \text{if } k \in C_j \\ 0 & \text{otherwise} \end{cases}$$

where β is a positive tuning parameter. Of course the time derivative $\mathcal{H}'_k(t)$ must be computed using a suitable discrete approximation (for example using the backward-difference operator and storing the previous value of the cumulative weight of each block).

The rationale for this approach can be summarised as follows: the cumulative weight of a block in a parasite chain grows linearly with rate proportional to the resources of the attacker, μ, whilst the main DAG will grow at the rate of the computing power of the rest of the network, λ, as illustrated in Figure 3.6. Therefore, the parasite chain will be heavily penalised by the First Order BRW.

Simulation results for the same instance of the DAG used for examples in Section 3.2 are shown in Figure 3.10. This figure consists of several traces, each for different values of the newly introduced parameter, β. Note that the trace corresponding to $\beta = 0$ is identical to that of Figure 3.4, because the new random walk model reduces to the original model in this case. These preliminary results suggest that the First Order BRW achieves its goal and effectively mitigates the chance of an attacker successfully double spending. Of course, the performance of the algorithm will start to deteriorate when the particle approaches a tip, as at this height, the cumulative weight on the main DAG grows at a lower rate. While we have not investigated mitigations for this issue in depth, it could be useful to modify the derivative term to decay with each jump forward.

3.5 Chapter Summary

In this chapter, we investigated the security of DAG-based DLTs under a well known double spending attack scenario called the parasite chain attack. Our analysis used a novel Markov chain model for the BRW tip selection algorithm (referred to as MCMC in the IOTA white paper [26]), and we validated this model using Monte Carlo simulations of random walks on randomly

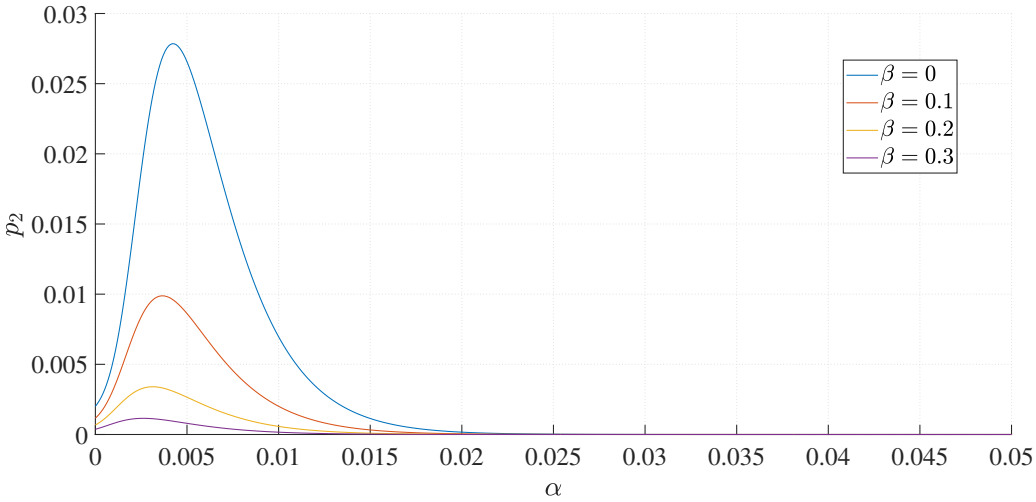


Figure 3.10: Probability of selecting an SPC tip: $\lambda = 15$, $\mu = 5$, $k = 1$, $T_{DS} = 120$.

generated DAG instances. We also presented an extension of the BRW algorithm called *First order*-BRW, which uses the first order time derivative of the cumulative weight. Our simulations demonstrate the modified algorithm's effectiveness at mitigating parasite chain attacks, compared to the standard BRW algorithm.

Chapter 4

Access Control for DAG-based DLTs without Proof of Work

***Abstract*— Distributed ledgers are not governed by any single entity, but rather by a collection of nodes that each make updates and contribute to maintaining the ledger. The degree of power that each node has in a given DLT network depends on the *access control* employed, so appropriate design of this component prevents tampering and misuse of the ledger by any individual or organisation with selfish or malicious motives. PoW access control, for example, allocates power to nodes based on how much computational power they can expend on appending blocks to the ledger, while PoR access control allocates power based on the reputation of each node. PoW access control is straightforward to implement for DAG-based distributed ledgers, but requires significant energy consumption which is not only wasteful and unethical but also prevents devices with contrained resources from participating. PoR access control overcomes these issues, but PoR solutions for blockchains do not generalise to DAGs. In this chapter, we present the first ever PoR access control for DAG-based ledgers which elimenates the need for PoW and all its associated issues.**

In this chapter, we propose an access control mechanism for DAG-based DLTs in order to guarantee full utilisation of network resources and fair access based on *reputation* (see Definition 1.3.1). Access control, as introduced in Section 1.3.3, enforces the chosen governance model of a DLT: for example, PoW access control ensures that those who can collectively consume the greatest amount of energy govern the state of the ledger. An alternative to PoW is sought after because it is highly inefficient and offers unsatisfactory performance [10]. Additionally, documented attacks on a DLT known as *Nano*[13] have demonstrated that PoW is not secure in the context of IoT networks with limited resources. Our access control algorithm offers the first IoT-friendly alternative, allowing block throughput to be controlled across the network in a manner that is fair and resistant to manipulation by malicious actors. Our solution represents a new design paradigm for DLTs, a networking-based approach which permits reputation-based access control to be integrated, in the place of PoW, into DAG-based ledgers for the first time. The contributions of this chapter are as follows.

- We model the access control problem for a class of DAG-based DLTs and provide a network model which takes into account limited buffer capacity and computational limitation of devices.

- We then present an access control algorithm for DAG-based distributed ledgers. The algo-

[13]Nano experienced a severe spam event which damaged their DLT network—`https://www.coindesk.com/nanos-network-flooded-spam-nodes-out-of-sync`. They have since announced primitive measures to prevent attacks on their PoW DAG which have similar motivation to our work—`https://senatusspqr.medium.com/nanos-latest-innovation-feeless-spam-resistance-f16130b13598`

rithm components include:

i) a scheduling algorithm which ensures fair access for all nodes according to their reputation and prevents honest nodes from being adversely affected by dishonest nodes seeking to take more than their fair share;

ii) a rate setting algorithm, inspired by transmission control protocol (TCP), which allows nodes to optimise their block issuing rate in a decentralised manner;

iii) A buffer management scheme to ensure that misbehaving actors can not cause buffers to overflow and compromise consistency for honest nodes.

- We provide extensive simulation results which demonstrate that the algorithm performs as intended, is robust to changes in the algorithm parameter choices, and is resilient against actors that deviate from the protocol by attempting to take more than their fair share of resources.

Remark: At the time of writing, this algorithm is under active development and its implementation and testing in IOTA's GoShimmer network is underway [56]. As such, additional features continue to be developed to enhance the security of the core algorithm. The model and algorithms presented in this chapter correspond to those in the original publications of this work [7, 8], whilst the following chapter will capture an updated version in line with current research and development. To the best of our knowledge, this algorithm is the first of its kind.

The remainder of this chapter is structured as follows. In Section 4.1 we provide an overview of relevant material and prior art from both the DLT and broader networking domain. In Section 4.2 we give a precise problem statement and state the requirements for our access control solution. In Section 4.3 we model the problem and provide the notation required to effectively describe and evaluate our solution. Section 4.4 presents the access control algorithm in detail, while Section 4.5 validates it and evaluates its efficacy through extensive simulations. Finally, in Section 4.6, we summarise the findings of the chapter.

4.1 Background and Related Research

This work lies at the boundary of DLT and the broader networking literature, including topics such as TCP, quality of service (QoS), gossip protocols and many more. We begin with the necessary DLT backdrop which also serves to motivate our problem. As we have already mentioned, to the best of our knowledge, our networking approach to DLT access control is completely new, so our review of literature only covers related technologies rather than comparable solutions.

4.1.1 Access Control for Distributed Ledgers

Access control for DLTs refers to how nodes determine who gets to write new data to the ledger in a secure and distributed manner. This is also known as Sybil protection, because it prevents so called Sybil attacks in which an attacker creates multiple identities in order to gain an illegitimate advantage [57]. PoW access control, used in blockchains such as Bitcoin [2], involves solving a computationally difficult puzzle to prove possession of computing resources to be allowed write to the ledger. PoW, however, consumes vast quantities of energy [38], which is unacceptable from an environmental standpoint, unfeasible for IoT devices and inevitably concentrates computing power in the hands of those who can access specialised hardware and cheap energy. This is the case in the Bitcoin network [58] where *miners* select which transactions to include in blocks during busy periods (typically based on which offer the highest transaction fees), providing an intrinsic mechanism for filtering transactions and preventing congestion. [59] proposes a credit-based system for adapting the difficulty of the PoW for certain nodes which behave well, aiming to make PoW more suitable for IoT scenarios. A similar approach involving adaptive PoW is examined in [60].

The access control algorithm presented here, on the other hand, accommodates a more general *reputation* model for Sybil protection mechanisms which do not require the use of computing resources, and are therefore more suitable for the IoT setting.

Reputation, as defined in Definition 1.3.1, is an input to our access control algorithm. Proof of Stake (PoS) is an example of reputation-based access control in which reputation is the currency owned by a node. In blockchains, PoS access control is typically implemented through some form of leader election in which a node or group of nodes becomes eligible to write a block each *round* through some randomised process [9, 39, 40]. Other examples of reputation-based access control include IOTA's *mana* system [44], delegated PoS and preconfigured node permissions as found in permissioned DLTs [61].

4.1.2 DAG-Based Distributed Ledgers

Our interest in DAG-based DLTs, rather than blockchains, lies in their ability to accommodate high block throughput with low latency, and their low barriers to participation which improve usability and decentralisation (see Chapter 2). Blocks may contain one [26] or more transactions [62]. Recall that in DAG-based ledgers, each new block can be cryptographically linked to more than one existing block, and many valid new blocks can be pointing to the same older block. The result is that many users of these ledgers can write blocks simultaneously, and therefore, there is no block throughput limit enforced by the ledger structure itself, as there must be in blockchains. Chapter 3 discusses the DAG structure and the security of these ledgers in more detail.

DAG-based ledgers traditionally rely on PoW for access control. It is difficult to incorporate reputation-based access control such as PoS due to the lack of structured rounds for *leader elections*. [63] attempts to avoid this issue by using a so called structured DAG for their PoS ledger which imposes strict limitation on when and how blocks can be added to the ledger. Due to our focus on IoT scenarios, we do not want to impose any such additional constraints, and hence, the rate at which nodes can issue and disseminate blocks must be controlled by some other mechanism. The access control algorithm we present here solves this problem by incorporating Sybil protection at the level of dissemination of blocks in the peer-to-peer network, regulating block throughput on a per-node basis. This new paradigm requires concepts from computer networking which we review next.

4.1.3 Networking Concepts

From the domain of replicated databases, [64] presents a *flow control* algorithm. The principle of what the authors of [64] aim to achieve is similar in nature to that of our access control algorithm, albeit in a trusted setting (without concern for adversarial behaviour). The flow control algorithm of [64] adaptively sets the update rate (block issuing rate) of nodes using a TCP-like algorithm, but the algorithm they present is designed for a controlled setting and no measures are taken to defend against nodes that issue updates at a higher rate than the protocol dictates. A simple First In First Out (FIFO) scheduler is assumed to be used in [64], and buffer overflows are used to signal congestion and reduce update rate. This presents an opportunity for malicious nodes to deflate the rate of others by simply setting their own rate too high. Additionally, trusted communication is used by nodes in [64] to explicitly negotiate fair update rates, which is also exploitable by malicious agents in the DLT setting.

QoS in packet switched networks is related to our problem because we wish to fairly regulate flows of blocks from different sources. Classic examples of QoS architectures include Diffserv [65] and Intserv [66], which offer course and fine-grained QoS respectively. Both of these architectures rely on a backbone of trusted routers which are assumed to follow the protocol. Conversely, in DLT networks, no other individual node can be trusted to provide reliable information, making these classical architectures unsuitable. However, some of the core principles from these architectures are still applicable to DLTs, for example, the use of packet schedulers. A router employing a fair scheduling algorithm, in contrast to a simple FIFO scheduler, protects the flows of honest nodes from congestion caused by misbehaving flows [67]. Fair schedulers have been proposed with varying levels of complexity, each trying to emulate generalised processor sharing (GPS) as closely as possible. Weighted fair queuing [68] provides a good approximation of GPS, but with significant computational overhead for routers, while simpler schedulers such as those based on Deficit Round Robin (DRR) [69, 70] provide lightweight and scalable alternatives.

Another feature of QoS networks and of IP networks in general which is relevant to our access

control problem is transmission control. TCP [71] typically involves a distributed Additive Increase Multiplicative Decrease (AIMD) algorithm to set the transmission rate: nodes additively increase their transmission rate until congestion occurs (or is pre-empted), as signalled by some feedback from the network; and multiplicatively decrease their transmission rate in response to this congestion. In most forms of TCP, congestion is signalled when an acknowledgement is not received for a packet, and some other variants are based on Random Early Detection and require Explicit Congestion Notifications (ECNs) [72]. All of these AIMD algorithms require feedback from other nodes, which leaves these protocols open to attack in the DLT setting, so none of them are applicable to our problem in their entirety.

Remark: The algorithm that we shall propose is designed to operate in adversarial environments where nodes can deviate from the recommended protocol (this is a baseline assumption under which DLTs are designed). The need for resilience to attacks is not typically considered in traditional networking applications such as those discussed above, which makes importing ideas from the traditional networking community difficult, and makes benchmarking our algorithm against similar work difficult, because it simply does not exist.

4.2 Problem Statement

We propose an access control algorithm for regulating block throughput, on a per-node basis, in a DAG-based DLT network with reputation-based Sybil protection. The goal of the algorithm is to allocate a portion of the network resources to each node proportional to their reputation and to prevent detrimental congestion. Blocks are issued by nodes and disseminated around the peer-to-peer network. Each node must validate all blocks, add them to its local copy of the ledger, and then run some consensus algorithm. We call these steps *writing*. Writing is the bottleneck at which congestion can occur, and the goal of our algorithm is to allocate constrained resources fairly at this bottleneck.

The specifics of writing will vary across DLT implementations and may even vary from node to node. For example, in certain DLTs some nodes may do the most computationally heavy tasks while other limited nodes, such as IoT devices, perform lighter tasks while writing. Severely constrained devices may operate simply as *users*, relying on trusted *nodes* for a reliable view of the ledger[14]. In Chapter 6, we consider how to orchestrate the interaction between users and nodes in an enterprise setting to ensure good network usability for all parties involved.

Our access control algorithm seeks to maximise the rate of dissemination of blocks, subject to the writing bottleneck, while minimising delays. The algorithm must also meet the requirements listed below. These requirements are described at a very high level here, and defined more precisely before presenting our access control algorithm in Section 4.4.

- *Consistency*: if a block issued by an honest node[15] is written by one honest node, it should eventually be written by all honest nodes.
- *Fairness in dissemination rate*: the dissemination rate of each node should be allocated fairly according to the node's reputation.
- *Fairness in latency*: for a given dissemination rate, relative to the node's reputation, a node's blocks should experience similar latency.
- *Security*: malicious nodes[16] should be unable to interfere with any of the above requirements.

Remark: we do not write of *fairness* here in any ideological sense of the word, but in a far more specific and relative sense. We assume that we have some agreed-upon measure of reputation (recall Definition 1.3.1) for each node in our system and we seek to provide fairness relative to this metric. One could, for example, create a DLT network in which only a small preselected group of wealthy individuals hold any reputation and all other participants hold no reputation. Our

[14] trinity.iota.org/nodes
[15] An honest node is a node that follows the proposed protocol.
[16] A malicious node is a node that issues blocks at a higher rate than that which the protocol dictates.

access control would then seek to allocate all resources to the wealthy individuals and none to anyone else—this could be viewed as entirely unfair from an ideological standpoint while perfectly satisfying our fairness requirements.

The physical limits of devices and the particular consensus algorithm employed determine the rate of the writing bottleneck. This in turn determines the maximum performance of the network (blocks/transactions per second, confirmation time, etc.), and our access control algorithm ensures that this maximum performance can be reached.

4.3 Model and Notations

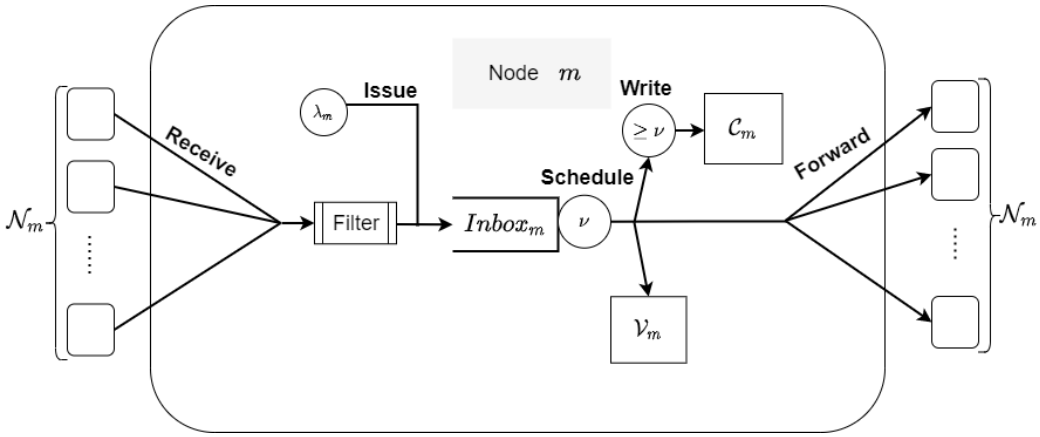


Figure 4.1: Model for a node m, indicating the actions taken by each node to process a block, namely, receiving, issuing, scheduling, writing and forwarding. λ_m denotes node m's block issuing rate, ν denotes its maximum scheduling rate, and with $\geq \nu$ denotes its writing rate which must be at least ν.

The model introduced in this section is illustrated in Figure 4.1, and the associated notation is summarised in Table 4.1 at the end of this section. The ledger is distributed over a set of nodes $\mathcal{M}$ in a peer-to-peer network, where a node m in $\mathcal{M}$ has a set of neighbours $\mathcal{N}_m \subset \mathcal{M}$ with which it communicates directly. Each node locally processes each block which they either issue themselves or receive from neighbouring nodes, as indicated in Figure 4.1. The subset of nodes which correctly follow the protocol, referred to as honest nodes, is denoted $\mathcal{M}^*$. The reputation distribution over the nodes (see Definition 1.3.1) is denoted *rep*, where rep_m denotes the reputation of node m. In the experiments presented here, *rep* is assumed not to vary with time.

Blocks are cryptographically signed, which links them to the identity of their issuer. The set of blocks that are visible to (i.e., scheduled by) node m (either issued by node m itself or received from neighbours) is denoted by $\mathcal{V}_m$. Each node additionally *confirms* a subset of these visible blocks and they are added the set of confirmed blocks $\mathcal{C}_m \subseteq \mathcal{V}_m$. Confirmation in DAGs is discussed in Chapter 2 and will be revisited in further detail in Chapter 5.

Definition 4.3.1 (Disseminated block)**.** We say that a block is *disseminated* when it has been received by all honest nodes, $\mathcal{M}^*$. The set of disseminated blocks, $\overline{\mathcal{V}}$, is defined as follows:

$$\overline{\mathcal{V}} = \bigcap_{m \in \mathcal{M}^*} \mathcal{V}_m \tag{4.1}$$

and $\overline{\mathcal{V}}^i$ denotes the subset of blocks in $\overline{\mathcal{V}}$ which were issued by node i.

Nodes in a given DLT have some constrained resource (e.g., computation or storage) that limits the rate at which they can process incoming blocks. The *writing work* of a block u is the work required from this constrained resource at each node in order to reach consensus and decide if u should be confirmed. We assume the expected writing work for u, denoted by $|u|$, to be known

in advance as it depends on known information such as the block payload type or size. Note that this modeling choice permits us to have a flexible algorithm where specific class of blocks can be prioritised, if needed. The expected writing work required to make decision on whether to confirm the blocks in a set A is denoted by $W(A)$, i.e.:

$$W(A) = \sum_{u \in A} |u| \tag{4.2}$$

and *writing power* corresponds to the rate at which this writing work is done. As writing power is the limited resource in our network model, the rate of dissemination of blocks must be measured in terms of the power required to write them.

Definition 4.3.2 (Dissemination rate)**.** The *dissemination rate*, DR, is the rate of dissemination of blocks, weighted by their work. This dissemination rate and the dissemination rate of node i's blocks, respectively, are defined as follows:

$$DR = \frac{\Delta W(\overline{\mathcal{V}})}{\Delta t} \tag{4.3}$$

$$DR_i = \frac{\Delta W(\overline{\mathcal{V}}^i)}{\Delta t} \tag{4.4}$$

where Δt is the time window over which we measure the dissemination rate.

Another important quantity for evaluating the performance of our access control algorithm is dissemination latency, which is defined as follows.

Definition 4.3.3 (Dissemination Latency)**.** The dissemination latency of a block is the time from when the block is issued to when it is added to $\overline{\mathcal{V}}$. In other words, dissemination latency is the random variable of the time it takes for a block to reach all honest nodes after it is issued.

Note that in order to achieve a consistent distributed ledger, all nodes must possess some minimum writing power to ensure that they can write blocks sufficiently quickly to keep up with the network's dissemination rate DR. For this reason we enforce a global writing power, ν, which all nodes must be able to achieve. This constraint is highlighted in Figure 4.1 by the circle beneath the word "Write".

The actions taken by nodes are highlighted in boldface on the node model diagram in Figure 4.1, and described as follows.

Receive

We assume reliable communication channels[17], hence a node m must receive all blocks sent by its neighbours $\mathcal{N}_m$. If communication channels are unreliable, no guarantees can be provided that a node will be able to acquire their fair share of resources, because other nodes must first be able to receive their blocks. These blocks are filtered at this point to remove duplicates and invalid blocks. A block can be considered invalid depending on the specific protocol employed: at a high level, this filtering criterion concerns signature and timestamp validation, and protection against denial of service attacks [73]. Filtered blocks are added to node m's inbox buffer, $Inbox_m$.

Issue

Nodes can additionally issue their own blocks, and these blocks are also added to the issuing node m's inbox buffer, $Inbox_m$. The rate at which node m issues blocks is denoted by λ_m, and this is controlled by node m using a *rate setting algorithm*. If all nodes wished to have as many blocks as possible written at all times, then a fair allocation of the writing power would permit each node to have an assured issuing rate, $\tilde{\lambda}_m$, defined as:

$$\tilde{\lambda}_m = \frac{\nu \cdot rep_m}{\sum_{i \in \mathcal{M}} rep_i}. \tag{4.5}$$

[17] Point-to-point connections are handled with TCP on a separate network layer, and hence can be considered reliable.

with units of work per second. Assuming a fair allocation of writing resources in the network can be achieved, node m can issue blocks at a rate less than or equal to $\tilde{\lambda}_m$, safe in the knowledge that these blocks will be written by all nodes without causing backlogs and delays, regardless of what rate other nodes issue blocks at. If a node wishes to issue blocks at a rate greater that $\tilde{\lambda}_m$, this must be done taking the issuing rate of other nodes into account so as to avoid excessive congestion. This latter observation motivates the need for the *rate setting* component of our access control algorithm to allow nodes to effectively use excess capacity.

To capture the varying demand across nodes to issue blocks, we define four modes of operation for nodes issuing blocks.

Definition 4.3.4 (Inactive)**.** A node m is said to be in *inactive* mode if it is not issuing any blocks, i.e., $\lambda_m = 0$.

Definition 4.3.5 (Content)**.** A node m is said to be in *content* mode if it is issuing blocks at a fixed rate $\lambda_m \leq \tilde{\lambda}_m$. This is modelled as a Poisson process with rate parameter λ_m, which is a standard model for arrival processes. There is no need for content nodes to regulate their issuing rate in response to network traffic because they are content to issue at a rate below their guaranteed minimum.

Definition 4.3.6 (Best-effort)**.** A node m is said to be in *best-effort* mode if it is issuing blocks at the highest rate possible under the current traffic conditions, without causing excessive congestion. This requires a node to use the rate setting algorithm, outlined in Section 4.4, to utilise unused network resources and adaptively set $\lambda_m > \tilde{\lambda}_m$. We assume that a leaky bucket regulator with rate λ_m is used to achieve the set issuing rate i.e. the issuing rate is deterministic, rather than Poisson.

Definition 4.3.7 (Malicious)**.** A node m is said to be in *malicious* mode if it is issuing blocks at a rate $\lambda_m \gg \tilde{\lambda}_m$, without concern for the congestion caused. We assume that a malicious node immediately writes and forwards its own blocks rather than including them in the scheduling steps.

Schedule

Blocks issued by node m itself and those received from neighbours are all added to $Inbox_m$, as described above. $Inbox_m^i$ denotes the blocks in $Inbox_m$ issued by node i. Blocks from $Inbox_m$ are then scheduled, added to the set of visible blocks, $\mathcal{V}_m$, and forwarded to neighbours. A fair scheduling algorithm [67] should be used to ensure that malicious agents issuing blocks at an excessive rate can not delay the blocks of honest nodes at the inbox buffers. This scheduling algorithm is discussed further in Section 4.4.

The scheduling process is deterministic, with rate ν, when there are blocks in the inbox to be scheduled. Note that this deterministic scheduling rate is in units of writing power and is imposed to ensure that the visible set of blocks can be written by all nodes having at least the minimum writing power ν.

Write

When a block has been scheduled, it is added to $\mathcal{V}_m$. The block is not yet considered confirmed at this point. Rather, the block must still satisfy the consensus rules of the DLT. The consensus protocol can vary between DLTs. For example, weight-based confirmation could be used, whereby the cumulative weight of the block must be computed to exceed some threshold. Alternatively, or indeed additionally, milestone-based confirmation can be employed, in which a voting algorithm such as [46] or [74] is used to achieve consensus on a milestone block which confirmes everything in its past cone. Each of these kinds of confirmation will be discussed in further detail and applied in Chapter 5.

The complexity of the aforementioned consensus protocols increases with the number of blocks involved, and varies with the type of blocks (for example, blocks containing sensor measurement data may be cheaper to reach consensus on than blocks with transaction payloads which transfer currency between accounts). The limit ν is a parameter of the DLT which is configured to ensure that any node with some minimum writing power can participate in the consensus.

Forward

After a block is scheduled by node m, it is forwarded to all neighbours $\mathcal{N}_m$ except for the neighbours from which the block has already been received (these are the only nodes that m can be sure already have this block). This is known as *flooding*, and while it is a highly inefficient use of communication resources, we defer any optimisation for the sake of robustness and simplicity of analysis of our access control solution. Duplicate blocks received as a result of flooding can easily be filtered out and do not affect our results here.

Table 4.1: Notation for node and network model.

$\mathcal{M}$	set of all nodes in the network
$\mathcal{M}^*$	set of all honest nodes
$\mathcal{N}_m$	set of nodes that are neighbours of node m
rep_m	reputation of node m
$\mathcal{V}_m$	set of blocks visible to node m
$\mathcal{C}_m$	set of confirmed blocks in node m's ledger
$\overline{\mathcal{V}}$	set of all disseminated blocks
$\overline{\mathcal{V}}^i$	set of disseminated blocks issued by node i
DR	dissemination rate (all blocks)
DR_i	dissemination rate (blocks issued by node i)
ν	global block writing power
λ_m	issuing rate of node m
$\tilde{\lambda}_m$	assured issuing rate of node m

4.3.1 Definition of Requirements

We now provide more precise definitions for each of the requirements stated in Section 4.2.

Definition 4.3.8 (Consistency)**.** Consider a finite time window $w \in \mathbf{R}^+$, and a finite offset $h \in \mathbf{R}^+$. At time $t + h$, if all blocks added to $\mathcal{V}_m$ for any node m within time $[t - w, t]$ are in $\overline{\mathcal{V}}$, the access control algorithm of this network is said to satisfy the consistency requirement.

The interpretation of this consistency requirement all nodes must eventually receive all blocks, which is essential if consensus on the ledger is to be achieved.

Definition 4.3.9 (Fairness in dissemination rate)**.** An access control algorithm satisfies the *fairness in dissemination rate* requirement if allocation of dissemination rate among nodes is max-min fair, weighted by each node's reputation. An allocation is max-min fair if an increase in any node's dissemination rate decreases the dissemination rate of another node m with equal or smaller *reputation-scaled* dissemination rate, DR_m/rep_m.

This fairness in dissemination rate requirement ensures that network resources are allocated to nodes based on their reputation so that each node gets fair access to issue blocks, and hence a fair vote. If this requirement is even approximately satisfied, then the ledger is resilient against Sybil attacks and can be safely deployed in public and adversarial settings.

Definition 4.3.10 (Fairness in dissemination latency)**.** We say that an access control algorithm satisfies the *fairness in dissemination latency* requirement if the expected[18] dissemination latency of a node's blocks is independent of its reputation, and increases with its reputation-scaled dissemination rate.

This fairness in dissemination latency requirement ensures that blocks belonging to honest nodes do not experience excessive delays. This requirement also ensures that nodes attempting to achieve a higher dissemination rate than they should be entitled to will experience high delays. Delay does not directly translate to a node's ability to contribute to consensus as dissemination rate does, so this is a soft requirement and approximate fairness in dissemination latency is sufficient.

[18]This property may also be defined in terms of maximum latency, rather than expected latency, depending on the requirements of the specific ledger.

Definition 4.3.11 (Security)**.** An access control algorithm satisfies the *security* requirement if the requirements defined in Definitions 4.3.8–4.3.10 are still satisfied in the presence of malicious actors.

This final requirement of security is essential for the public DLT environment in which some nodes (malicious actors) may try to gain an unfair advantage by deviating from the protocol. The security requirement ensures that this can not happen.

4.4 Access Control Algorithm

The relevant notation and definitions are now in place, and we can now present our solution which consists of three core components, namely a scheduler, a rate setter and a buffer manager.

Scheduler: The scheduling component aims to ensure that blocks issued by honest nodes do not experience delays due to congestion caused by dishonest nodes. To this end, in the presence of congestion, blocks should be scheduled at a rate proportional to the reputation of the node that issued the blocks.

Rate setter: The rate setting component seeks to allow *best-effort* nodes (see Definition 4.3.6) to issue at a rate above their assured rate, $\tilde{\lambda}_m$, without causing excessive congestion and large delays which could cause a violation of the consistency requirement.

Buffer manager: The buffer management component decides when to drop blocks to protect honest nodes' blocks from being dropped due to congestion caused by malicious nodes. Under normal network operation, the rate setting component should prevent the need for any dropped blocks, but in the event of a malicious agent issuing blocks at an excessively high rate, buffer management can ensure that only this malicious node's blocks are dropped and the buffer does not reach its physical capacity.

4.4.1 Scheduler

Nodes in our setting are capable of more complex and customised behaviour than a typical router in a packet-switched network, but our scheduler must still be efficient and scalable due to the potentially large number of nodes requiring differentiated treatment. It is estimated that over 10,000 nodes operate on the Bitcoin network[19], and we expect that an even greater number of nodes are likely to be present in the IoT setting. We therefore adopt an efficient and scalable scheduler based on Deficit Round Robin (DRR), with modifications to deal with particular features of a DLT network, namely high variance in reputation among nodes and potentially bursty traffic. [69]. The standard Linux implementation of the DRR-based scheduler used in [75] permits up to 65,535 separate queues, which demonstrates the scalability of these methods.

DRR-based scheduling algorithms are very simple: each flow of packets (blocks) is visited in a round robin cycle, and deficit is assigned to the flow. Deficit can be thought of as credits to schedule packets, where sufficient credits must be accrued by a flow in order to have a packet scheduled. Our scheduling algorithm, DRR−, is presented in Algorithm 1. Node m maintains a deficit counter, DC_m^i, for each node i in $\mathcal{M}$. Each node in $\mathcal{M}$ is considered in a round robin cycle, one after another. Regardless of whether $Inbox_m^i$ has any blocks in it, DC_m^i is incremented by a quantum, Q_i, which is proportional to rep_i, up to a maximum $DC_{\max}$. The unit of deficit here is that of writing work, and deficit $|u|$ must be spent from DC_m^i in order to schedule a block u from $Inbox_m^i$. Blocks in $Inbox_m^i$ for each i are scheduled in FIFO order. The parameter $DC_{\max}$ should be chosen to be higher than the maximum work required to write a single block, and such that $Q_i \ll DC_{\max}$ for all nodes.

In standard DRR [69], a flow must be backlogged (packets from this flow must be waiting in the queue) in order to gain deficit. This feature of DRR presents problems for bursty traffic, because queues may periodically empty between bursts. This is problematic in DLTs because blocks must traverse multihop paths in the P2P network and the dynamics of these paths result

[19] https://bitnodes.io/.

in bursty arrivals at nodes' inbox queues. In our setting, higher reputation nodes can issue at a higher rate, meaning that queues are backlogged with their blocks more often even when the arrival of their blocks is bursty. Conversely, low reputation nodes issue at a lower rate, and queues may be completely emptied of their blocks between bursts. This gives higher reputation nodes an advantage in the standard DRR scheduler. DRR++ is a modification of the standard DRR scheduler designed to ensure low delays for so-called latency-critical flows in the presence of bursty arrivals [70]. DRR++ performs well for a small portion of latency-critical flows but becomes more comparable to standard DRR when all flows are deemed latency-critical, as we require in the DLT setting.

DRR−, on the other hand, can accommodate bursty traffic from all nodes. The principle behind DRR++ is essentially to allow latency-critical flows to go into negative deficit so that they can be scheduled rapidly. Our approach instead allows flows to gain deficit up to some limit, even when the flow is not backlogged, saving rather than going in to debt. This allows bursty traffic while maintaining efficiency because nodes with maximum saved deficit (indicating that they are inactive) do not need to be visited by the scheduler. The scheduler presented in [76] also uses a concept of deficit savings to accommodate bursty flows. However, [76] does not permit differentiated treatment of flows based on reputation and their analysis focuses primarily on active queue management to prevent bufferbloat [77].

Algorithm 1 DRR− Scheduler

Initialise:
$DC_m^i \leftarrow 0, \quad \forall i \in \mathcal{M}$

Repeat for $i \in \mathcal{M}$ in a round robin cycle:
if $DC_m^i < DC_{\max}$ **then**
  $DC_m^i \leftarrow DC_m^i + Q_i$
end if
while $\left|Inbox_m^i\right| > 0$ **do**
  $u \leftarrow$ oldest block in $Inbox_m^i$
  if $DC_m^i \geq |u|$ **then**
    Schedule u
    $DC_m^i \leftarrow DC_m^i - |u|$
    Wait $\frac{|u|}{\nu}$ seconds
  else
    break
  end if
end while

Table 4.2: Scheduling algorithm parameters.

ν	scheduling rate
Q_i	quantum added to DC_j^i, $\forall j$ in each round ($\propto rep_i$)
$DC_{\max}$	maximum deficit for an empty queue

4.4.2 Rate Setter

If all nodes always had blocks to issue, the problem of rate setting would be very straightforward: nodes could simply operate in *content* mode, at a fixed, assured rate, $\tilde{\lambda}_m$ (see Definition 4.3.5). The scheduling algorithm ensures that this rate is enforceable and that increasing delays or dropped blocks are only experienced by misbehaving nodes. However, it is highly unlikely that all nodes will always have blocks to issue, and we would like *best-effort* nodes to better utilise network resources, without causing excessive congestion and violating requirements.

Our rate setting algorithm, for *best-effort* nodes, is inspired by TCP — each node uses AIMD (see Section 4.1) rules to update their issuing rate in response to congestion events [78]. However, in the trustless DLT setting, the traditional means of responding to congestion is compromised. For example, malicious nodes could attempt to deflate the issuing rate of their neighbours by not

sending acknowledgements, or sending illegitimate congestion notifications. We observe, however, that in distributed ledgers, all block traffic passes through all nodes, contrary to traffic typically found in packet switched networks and other traditional network architectures. Under these conditions, local congestion at a node indicates congestion elsewhere in the network. This observation is crucial, as it presents an opportunity for an access control algorithm based entirely on local traffic and without the need for additional (potentially corruptible) interactions between nodes.

Recall that when a node m issues a block, it is added to its inbox buffer to be scheduled. Node m's own blocks in its inbox, $Inbox_m^m$, are then scheduled at a rate which depends on the other traffic present in the buffer. We observe that the length of $Inbox_m^m$ gives an estimate of congestion in node m's traffic, not only at its own inbox buffer but at $Inbox_i^m$ for all nodes i in $\mathcal{M}^*$, within some network delay.

Algorithm 2 outlines the AIMD rules used by each node to set their issuing rate, and the parameters of the rate setting algorithm are outlined in Table 4.3. Each node sets their own local additive-increase parameter based on the global increase rate A, and their reputation. Specifically, each node sets their local increase parameter as follows:

$$\alpha_m \leftarrow A \cdot \frac{rep_m}{\sum_i rep_i}. \tag{4.6}$$

An appropriate choice of A ensures a conservative global increase rate which does not cause problems even when many nodes increase their rate simultaneously. Updates are made to the issuing rate each time a block is scheduled, at a rate proportional to the writing work of the block, $|u|$, which allows the rate setter to accommodate variable block types and sizes. Nodes wait τ seconds after a multiplicative decrease, during which there are no further updates made, to allow the reduced rate to take effect and prevent multiple successive decreases. Waiting after decreases is common in implementations of AIMD algorithms, such as sliding window flow control in TCP [71]. The rate is updated each time a block is scheduled. At each update, node m checks the work $|Inbox_m^m|$ and responds with a multiplicative decrease if this is above a threshold, $W \cdot rep_m$, where W is a global parameter. If $|Inbox_m^m|$ is below this threshold, m's issuing rate is incremented by its local increase parameter α_m. $|Inbox_m^m|$ is measured as an exponential moving average with samples taken each time a block is scheduled.

Table 4.3: Rate setting algorithm parameters.

A	global additive increase parameter
β	global multiplicative decrease parameter
τ	wait time parameter
W	inbox work threshold

Algorithm 2 AIMD Rate Setter (Best-effort Mode)

Repeat each time a block u is scheduled:
if $|Inbox_m^m| > W \cdot rep_m$ **then**
    $\lambda_m \leftarrow \lambda_m \cdot \beta$
    Pause issuing and rate setting for τ seconds
else
    $\lambda_m \leftarrow \lambda_m + \alpha_m \cdot |u|$
end if

4.4.3 Buffer Manager

Nodes' inbox buffers do not have infinite capacity, and buffer capacity can be particularly limited in the case of IoT devices. Even if the capacity of buffers could be made arbitrarily large, an excessively full buffer could result in large delays [77]. Our buffer management seeks to drop blocks fairly (with respect to issuing node's reputation) whenever the buffer exceeds a certain size. The objective of our buffer management differs from than that of typical active queue management (AQM) systems. AQM is generally used to regularly drop packets and generate explicit congestion notifications (ECNs), forming a key component of how congestion is detected for rate setting. An

early example of this kind of AQM can be found in [72], and it is also a component of the schedulers discussed above [76, 75]. However, dropped blocks or ECNs are not used to detect congestion in our setting, so our buffer management does not play a direct role in rate setting. Rather, buffer management plays the important role of ensuring that malicious nodes that do not abide by the rate setting rules can not fill the buffers and cause honest blocks to be dropped. Our rate setting algorithm prevents excessive congestion under normal operation and buffer management should only take effect in the presence of malicious behaviour.

The simple buffer management rule employed in this chapter is stated in Algorithm 3. This buffer management is equivalent to the *longest queue drop* scheme proposed in [79]. The parameter $W_{\max}$ is the maximum work in the inbox buffer after which blocks should be dropped. We assume that each block has some maximum size in memory and requires some minimum work, so $W_{\max}$ bounds the memory needed. Provided the buffer memory is specified to be larger than this worst case memory, the buffer management will prevent the buffer from overflowing.

If the work in the buffer exceeds $W_{\max}$, the node with the greatest amount of work in the buffer, relative to its reputation, is identified, and the first received block is dropped, as outlined in Algorithm 3. This buffer management strategy ensures that consistency is preserved for honest nodes, because the work from honest nodes in the buffer should remain modest and hence their honest blocks will not be dropped — the scheduler ensures that this is the case as it will continue to schedule fairly without regard for large influxes of blocks from malicious nodes.

Algorithm 3 Buffer Manager

1: **while** $|Inbox_m| > W_{\max}$ **do**
2: $\quad d \leftarrow \arg\max_{i \in \mathcal{M}} \frac{|Inbox_m^i|}{rep_i}$
3: $\quad$ Drop block from head of $Inbox_m^d$
4: **end while**

4.5 Simulations

We now present simulations to evaluate the efficacy of our approach. The main focus of our evaluation is to verify that the requirements have been met, as these guarantee that the resources available to the nodes are optimally utilised both fairly and securely. To this end, we provide simulations for both honest and malicious environments, then demonstrate the robustness of our solution to parameter choices. After that, we demonstrate how our algorithm can accommodate different node types and their differing usage of the ledger in IoT settings. Finally, we benchmark our work against PoW access control, which represents the current state of the art access control solution for DAG-based DLTs.

In terms of evaluating performance, our results show that we can achieve close to 100% utilisation of node resources up to the limit, ν, set to ensure all nodes can keep up with the rate of newly issued blocks. The number of blocks per second that a DLT employing this access control can handle will therefore depend on the resources available to nodes and the consensus algorithm employed which together will determine the achievable rate ν.

The results in this section are produced with a Python simulator for DAG-based distributed ledgers[20]. We consider a network of 50 nodes, each storing a copy of the ledger, and each with 4 randomly selected neighbours, resulting in a random 4-regular topology of the peer-to-peer network. The mean propagation delay of each communication channel between neighbours is chosen uniformly at random between 50 ms and 150 ms, and the delay for each block on these channels is normally distributed around this average with standard deviation 20 ms. Node reputation is computed according to real data, that is, the number of blocks issued by each account in the IOTA network and follows a Zipf distribution[21] with exponent 0.9. Simulation results are averaged over 20 Monte Carlo simulations, each 180 seconds simulation time, where the simulator is stepped in

[20] Source code available at https://github.com/cyberphysic4l/DLTCongestionControl/tree/iotj.

[21] Wealth has also been shown to follow similar distributions, so this model is also well suited to reputation systems derived from wealth, i.e., PoS [45].

increments of 10 milliseconds. We assume that each node has buffer capacity greater than the parameter $W_{\max}$ specified for the buffer management, so no buffer overflows occur.

4.5.1 Honest Environment

The first set of simulations is in an honest environment, where each node is operating in one of the three honest modes: inactive, content, or best-effort (see Definitions 4.3.4–4.3.6). We evaluate whether dissemination rate is maximised, dissemination latency is minimised, and the first three requirements are met, namely consistency, fairness in dissemination rate and fairness in dissemination latency (see Definitions 4.3.8–4.3.10). The distribution of reputation and operating mode is illustrated in Figure 4.2. The global block writing rate is $\nu = 50$ units of work per second and each block requires one unit of work unless otherwise specified, i.e., $|u| = 1$ for all blocks u. The parameters of the access control algorithm for this set of simulations, given in Table 4.4, are chosen experimentally.

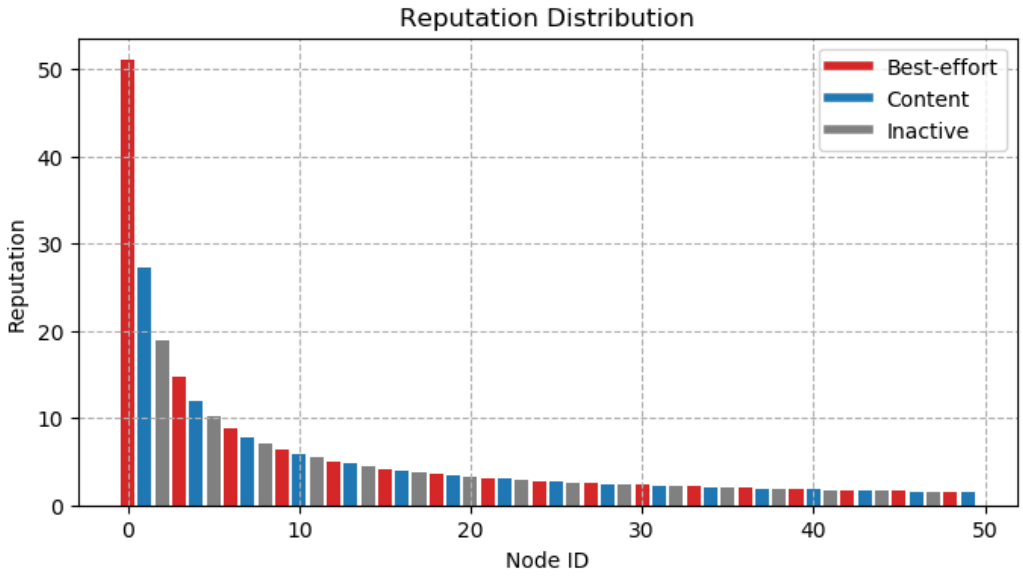


Figure 4.2: Reputation distribution follows a Zipf distribution with exponent 0.9. Nodes are content, best-effort, or inactive as indicated by each bar's colour.

Table 4.4: Access control algorithm parameters.

Scheduler			**Rate Setter**				**Buffer Man.**
ν	Q_i	$DC_{\max}$	A	β	τ	W	$W_{\max}$
50	$\frac{rep_i}{\sum rep}$	1	0.075	0.7	2	2	200

Figure 4.3 shows the overall dissemination rate, DR, (see Definition 4.3.2) for this set of simulations alongside the mean dissemination latency (see Definition 4.3.3) over all disseminated blocks. DR is shown as a percentage of ν because ν is the maximum rate blocks can be disseminated. We observe that the dissemination rate converges to a value close to 100% and that the mean dissemination latency converges to a steady state, in this case around 5 seconds. The mean dissemination latency depends heavily on the network diameter and delays associated with each hop, and the dissemination rate and latency convergence values depend on the rate setting parameters as demonstrated below.

The consistency requirement is demonstrated by Figure 4.4 which shows the maximum time in transit. The time in transit can be measured for a block that is not yet disseminated, and is the time spent in the system, i.e., the time from when it was issued to the present. This value converges to a finite value in Figure 4.4, demonstrating that consistency is achieved in the honest environment. Also note that no blocks were dropped by the buffer manager in these simulations,

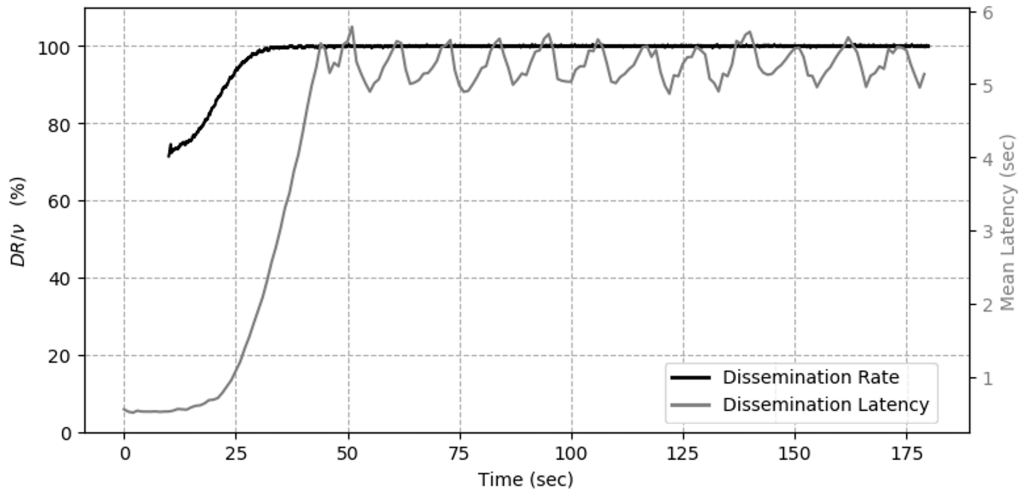


Figure 4.3: Dissemination rate and mean dissemination latency over all blocks.

in line with research in [79]. We can observe small oscillations in these measurements which can be explained by the AIMD rate setter employed by best effort nodes which periodically increases the rate of these nodes until congestion occurs, and then decreases the rate.

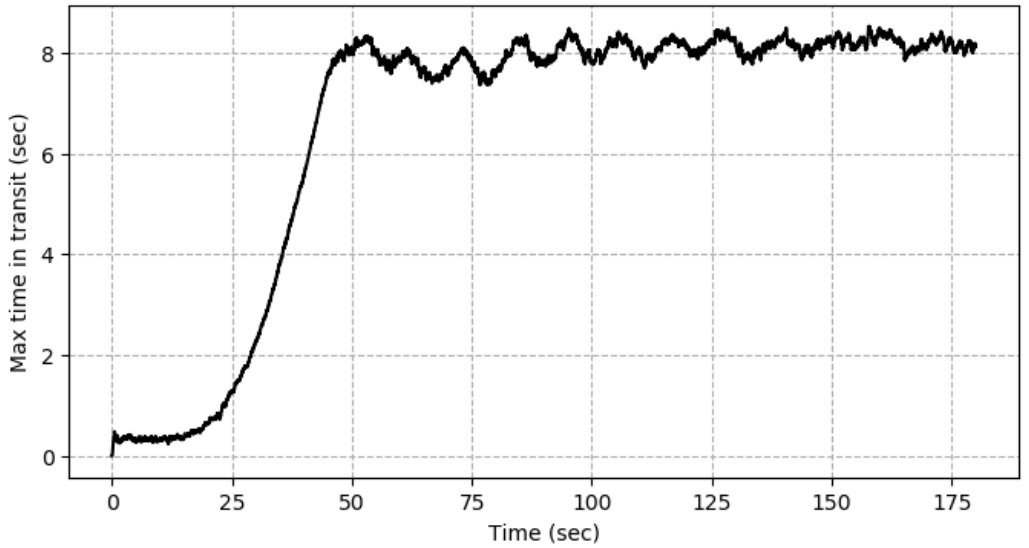


Figure 4.4: Maximum time since issue for all undisseminated blocks, demonstrating that consistency is achieved.

Fairness in dissemination rate is demonstrated by Figures 4.5. The upper subplots show the dissemination rate of each node, and the lower plot shows this dissemination rate scaled by each node's reputation. Best-effort nodes are plotted in red, and content nodes in blue, with the thickness of each trace proportional to the reputation of the relevant node, i.e., higher reputation nodes' rates are plotted with thicker lines. The bottom plot shows a particularly crucial result, namely it demonstrates that each node gets fair access to write to the ledger according to its reputation. In other words it implements Sybil protection, and we will show below that malicious actors can not tamper with this by deviating from the protocol.

To further demonstrate fairness in dissemination rate, Figure 4.6 shows results from simulations

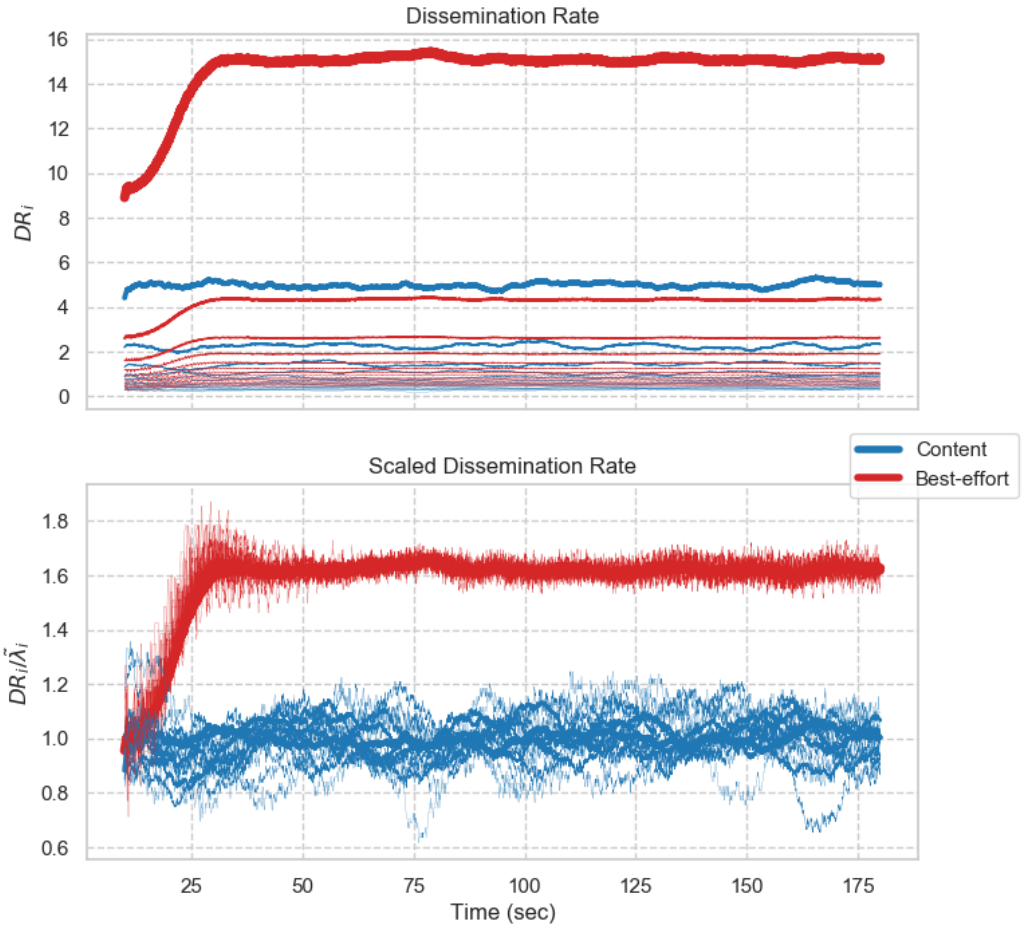


Figure 4.5: Dissemination rate and scaled dissemination rate of each node. The bottom plot of scaled dissemination rate demonstrates that fairness in dissemination rate is achieved.

in which the highest reputation content node switches to best-effort mode after 90 seconds. This node's dissemination rate is shown in purple and we can clearly see that all best-effort nodes adapt their issuing rates to maintain fairness under the new traffic conditions.

Fairness in dissemination latency is demonstrated by Figure 4.7, which shows the cumulative density function of dissemination latency across blocks issued by each node. Figure 4.7 includes a comparison between the DRR− scheduler and a standard DRR scheduler to demonstrate the improvements made by our design. The same convention for colour and line thickness as Figure 4.5 is used here. Clearly, fairness in dissemination latency is only approximately achieved, with lower reputation nodes experiencing higher latency than higher reputation nodes. However, our DRR− scheduler a significant improvement over the standard DRR scheduler. Low reputation nodes receive slightly unfair treatment from the scheduler because their blocks are emptied more often from inboxes since they have a lower issuing rate and hence less backlog. In standard DRR, deficit can not be gained when an inbox is empty, so lower reputation nodes gain less deficit and inboxes become disproportionately backlogged with their blocks, resulting in higher delays. This problem is ameliorated in our scheduler (DRR−) by sometimes allowing deficit to be gained with an empty inbox.

4.5.2 Adversarial Environment

In order to test the security requirement, malicious nodes must be introduced to the simulation while consistency, fairness in dissemination rate and fairness in dissemination latency are not

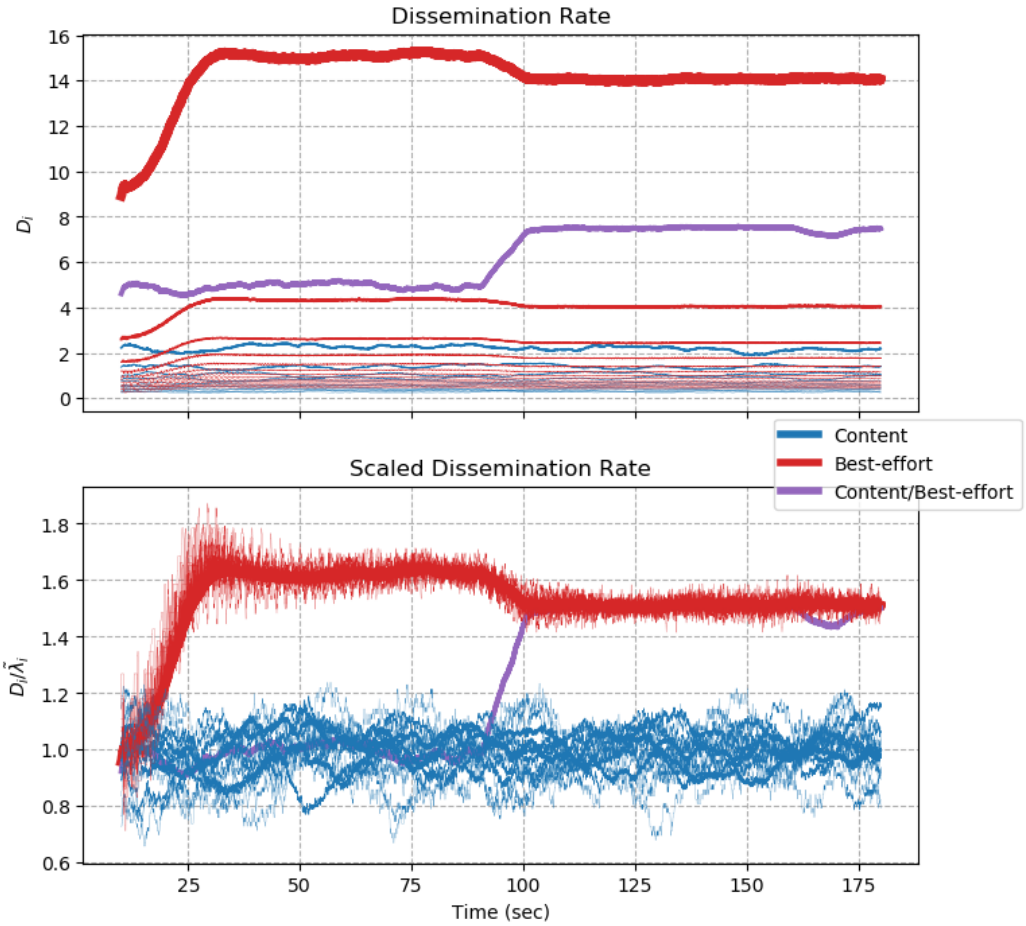


Figure 4.6: Dissemination rate and scaled dissemination rate of each node. The highest reputation content node (purple) switches to best-effort after 90 seconds and other best-effort nodes must adapt their rates.

compromised for honest nodes. We focus our attention on the malicious behaviour defined in Definition 4.3.7 as we can not anticipate all potential attack strategies. This attack serves to show that a node can not simply inflate their dissemination rate beyond what its reputation allows. Retaining the above network topology and reputation distribution, we introduce malicious nodes as illustrated in Figure 4.8, where we simply make every fourth node malicious.

Figure 4.9 shows the maximum time spent in the system for blocks issued by honest nodes. This measurement considers all undisseminated blocks in the system at each time step. This appears to converge as before, indicating that consistency is still achieved.

Figure 4.10 shows the dissemination rates and scaled dissemination rates for this set of simulations. Clearly, fairness is still achieved for honest nodes. The dissemination rate of the malicious nodes initially begins to converge to the max-min fair value also, but when they begin to cause excessive congestion, the buffer management component begins dropping malicious blocks and the malicious nodes' dissemination rate falls to zero.

Figure 4.11 demonstrates that approximate latency fairness is still achieved for the honest nodes but that malicious nodes experience far higher latency. Only the DRR− scheduler is displayed in this case as there is no further need to compare with standard DRR as in Figure 4.7.

Note that 12 of the 50 nodes simulated in this section issue at a greater rate than they should

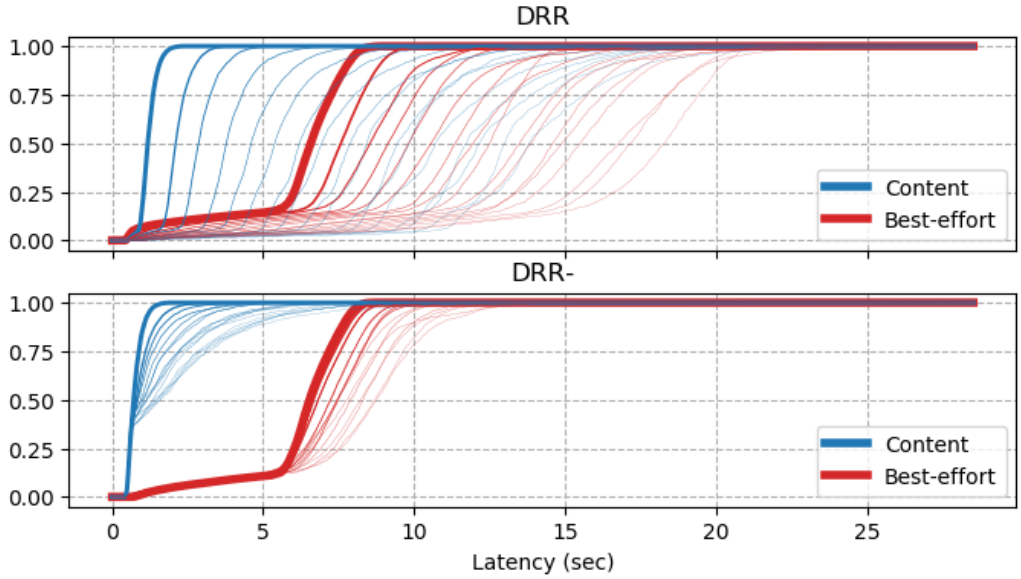


Figure 4.7: Cumulative distribution of dissemination latency for each node for DRR scheduler and DRR− scheduler. It is shown that only approximate fairness in dissemination latency is achieved, but DRR− performs far better than standard DRR in this respect.

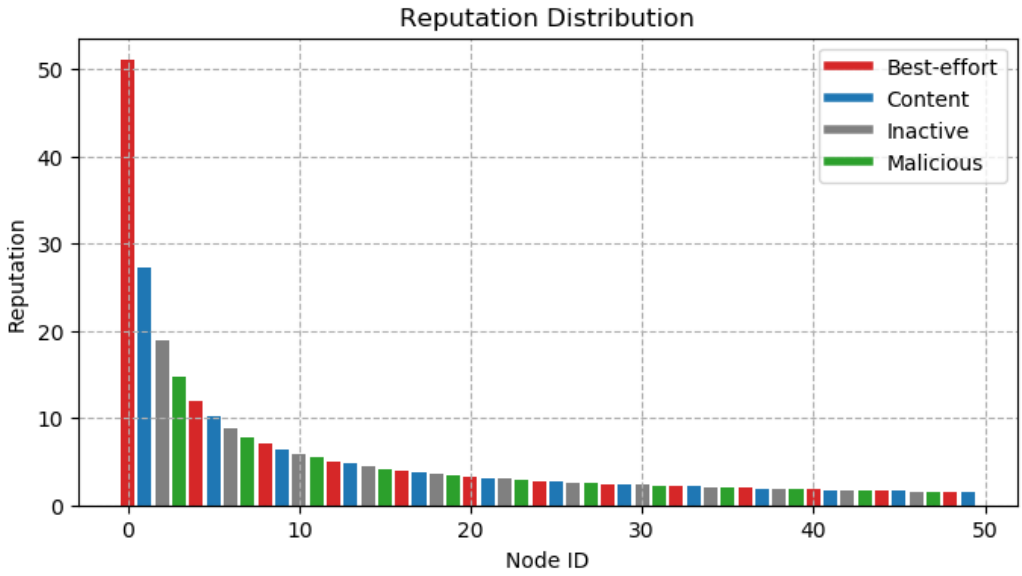


Figure 4.8: Reputation distribution following a Zipf distribution with exponent 0.9. Nodes are content, best-effort, inactive or malicious as indicated by the colour of each bar.

be allowed, and yet the fairness properties remain well satisfied. This is due to the fact that all an honest node requires in order to schedule all blocks at a fair rate is to receive the blocks in the first place. This means that the access could theoretically tolerate an arbitrary number of malicious nodes of this kind. In order to break the fairness requirement for an honest node, an attacker would need to prevent their blocks from reaching the rest of the network by ensuring they are not connected to any honest neighbours. This kind of attack is known as an eclipse attack, and can be prevented through measures at the level of the peer-to-peer network formation. Attacks of this kind on the network topology are beyond the scope of this work.

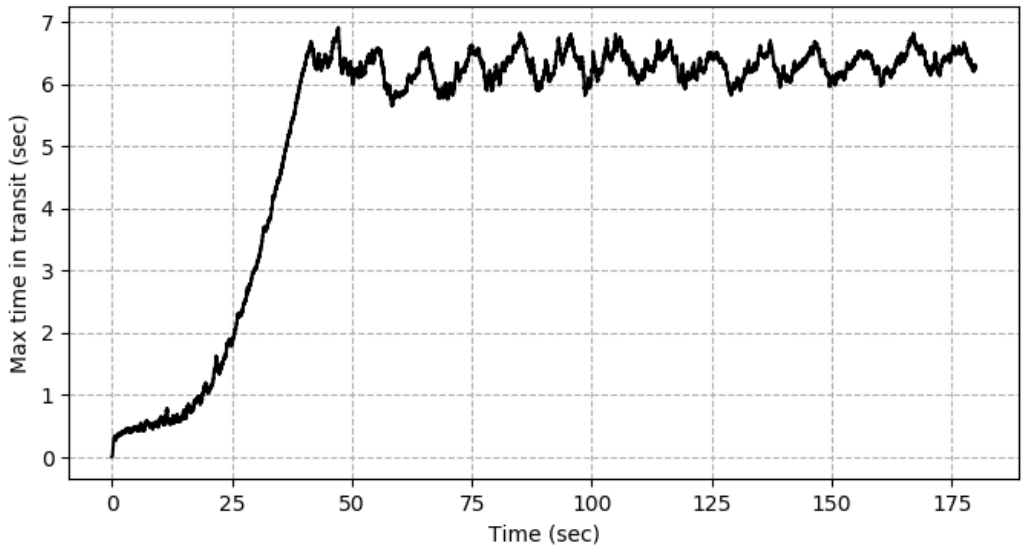


Figure 4.9: Maximum time since issue for undisseminated blocks issued by honest nodes.

4.5.3 Sensitivity Analysis

We now demonstrate how the network responds to tuning of the rate setting parameters. We focus particularly on the increase parameter A, the decrease parameter β, the work threshold W, and the total number of nodes $|\mathcal{M}|$. The wait time τ should simply be chosen long enough so that successive decreases do not falsely occur after a congestion event.

First, consider the increase parameter A. Beginning with the simulation parameters given in Table 4.4, we demonstrate the effect of increasing and decreasing A in Figure 4.12. It is shown that increasing A results in faster convergence to the equilibrium dissemination rate and mean dissemination latency, but that it has little impact on the equilibrium expected values.

Next, we analyse the impact of the decrease parameter β. Similar to the last experiment, we adjust β while fixing the other simulation parameters listed in Table 4.4. Figure 4.13 shows the corresponding results. A lower decrease parameter causes the issuing rate to decrease more drastically at congestion events. Thus we observe more oscillation in the mean latency, indicating that queue lengths oscillate more. Also we observe a very slight decrease in dissemination rate as β is decreased.

Finally, we demonstrate the impact of the work threshold W. This threshold corresponds to the level of backlog which is considered congestion, so this parameter has a significant impact on the queue lengths and hence the latency. This is illustrated in Figure 4.14, in which W is changed while keeping all other simulation parameters from Table 4.4 fixed. We observe that there is a very clear impact on the mean latency resulting from this choice, and less so from different choices of A and β. The significance of W is directly linked to the equilibrium level of congestion at which the rate setting responds, while A and β primarily determine how quickly the system reaches that equilibrium, and how aggressively the rate setting reacts to each congestion event, respectively. Note that if we decrease the threshold W too much, noise in the measurements of the inbox can become significant and fairness of the rate setting can be compromised. The lower limit of W to minimise latency while preserving fairness should be determined experimentally for deployment in a real network.

The rate setting parameters here are designed to scale with the network, so new nodes joining should not require re-tuning of parameters. Figure 4.15 presents dissemination rate and delay as the number of nodes is changed, with all other access control algorithm parameters remaining unchanged. The total reputation as given in Figure 4.2 is conserved, but is redistributed among

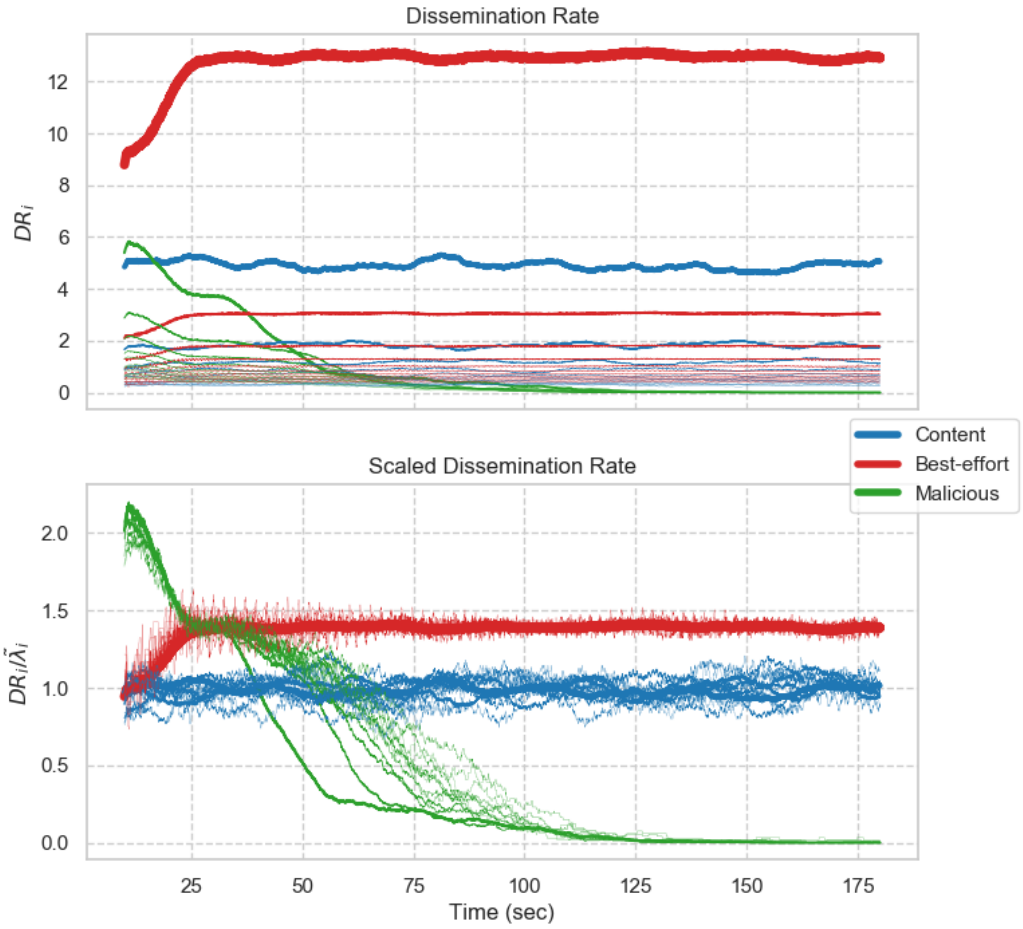


Figure 4.10: Dissemination rate and scaled dissemination rate for each node. The bottom plot of scaled dissemination rate demonstrates that fairness in dissemination rate is achieved for honest nodes, while malicious nodes are penalised by the buffer management and experience lower dissemination rates.

nodes to retain the Zipf distribution as nodes join and leave the network. It is clear that the algorithm performs well without requiring retuning of parameters as the network scales, although delay is increased slightly as the diameter of the network increases.

Deployment of our algorithm in a real DLT network (IOTA's GoShimmer network [56]) with varying number of nodes, real traffic and a range of network topologies will allow us to further verify the robustness of our parameter choices. Parameters relating to inbox lengths and scheduler parameters will require consideration of the buffer capacity available to nodes and the number of nodes in the network, which will also be examined when the algorithm is deployed in our test network.

4.5.4 IoT Devices and Variable Block Work

For simplicity of presentation, all blocks in the simulations presented so far require equal work to write and are therefore treated equally by the access control. In the IoT setting, blocks may contain varying payloads, such as sensor readings, machine-to-machine micro-payments, or a range of application-specific data, all of which require different levels of work to write. Contrary to IoT data blocks, we expect that blocks which transfer currency will typically require more work due to processes such as validity checks and updating account balances. In the following set of simulations, we demonstrate that our algorithm can handle variable block work requirements, and that lower

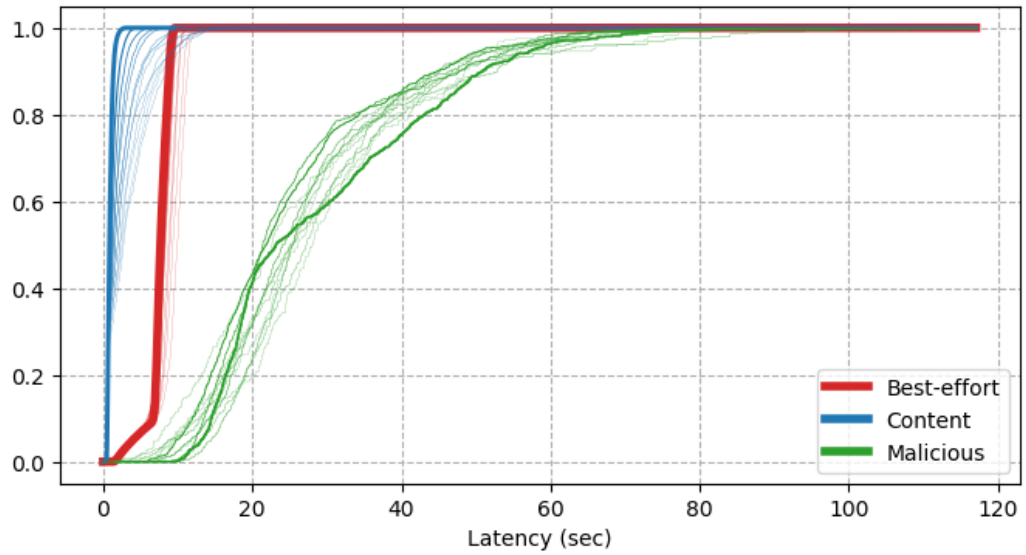


Figure 4.11: Cumulative distribution of dissemination latency for each node. Malicious nodes are shown to experience higher latency, while approximate fairness in dissemination latency is retained for honest nodes.

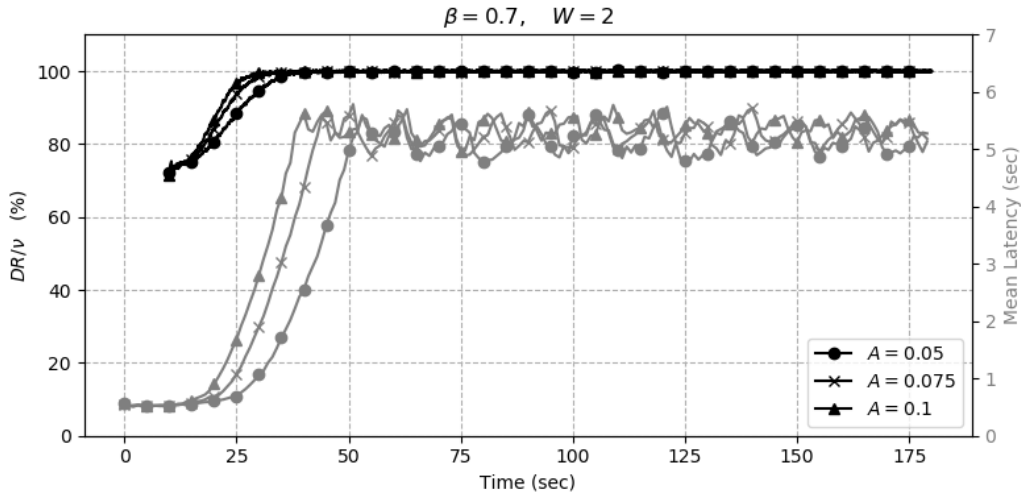


Figure 4.12: Combined dissemination rate as a percentage of ν, and mean dissemination latency, changing the additive increase parameter A.

work blocks actually experience lower latency, giving an advantage to IoT devices issuing such blocks. In order to illustrate the impact of varying block work, consider two representative node types.

- *Value node*: all blocks are value transfers and require one unit of work.
- *IoT node*: blocks contain an array of data types and require random work uniformly distributed on the interval $[0.25, 0.75]$.

Consider the same reputation distribution and operation modes as illustrated in Figure 4.2 with all even numbered nodes as value nodes and all odd number nodes as IoT nodes (note that nodes are numbered from 0 to 49). The access control parameters are as given in Table 4.4.

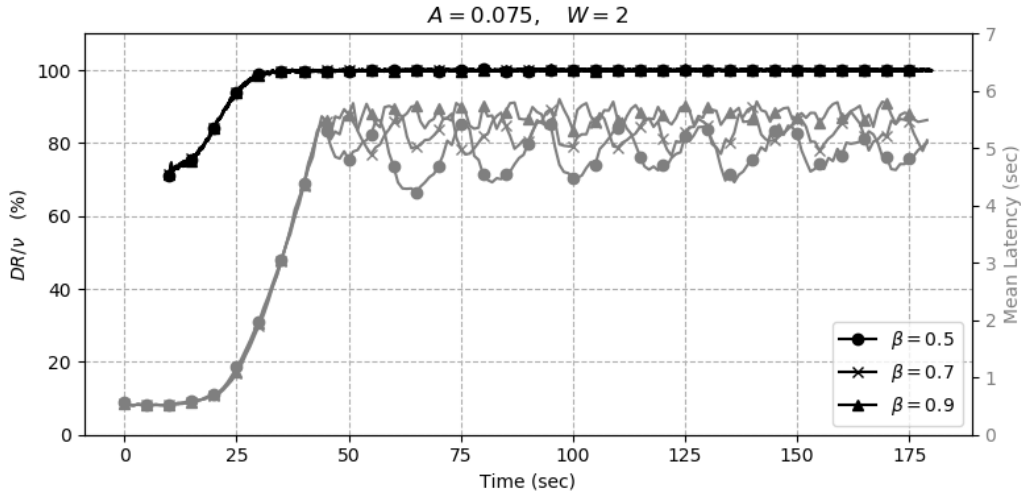


Figure 4.13: Combined dissemination rate as a percentage of ν, and mean latency, changing the multiplicative decrease parameter β.

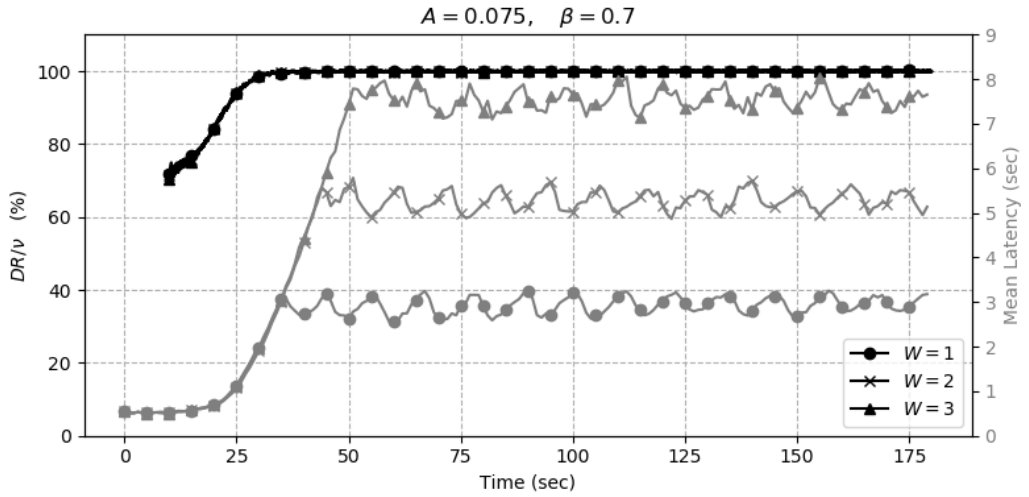


Figure 4.14: Combined dissemination rate as a percentage of ν, and mean latency, changing the work threshold parameter W.

Figure 4.16 shows that fairness is still achieved with this combination of IoT nodes and value nodes. Figure 4.17 shows the dissemination latency for each node. Approximate fairness in dissemination latency is still achieved, and more notably, IoT nodes have lower latency compared with the equivalent nodes in earlier simulations.

4.5.5 Comparison to Proof of Work Access Control

At the time of invention of the present access control algorithm, PoW access control was the state-of-the-art for DAG-based DLTs. PoW access control for DAGs is very straightforward: the difficulty of the PoW is set for the protocol and this determines how rapidly a node can create blocks with a valid PoW puzzle solved given its available computing resources. Blocks are simply scheduled in FIFO order. The PoW difficulty should be set such that writing resources are well utilised, but nodes with scarce writing resources can write all incoming blocks to their ledger without becoming congested. However, if the PoW difficulty is set too high, writing resources will be underutilised and dissemination rate will not be maximised. On the other hand, if it set too

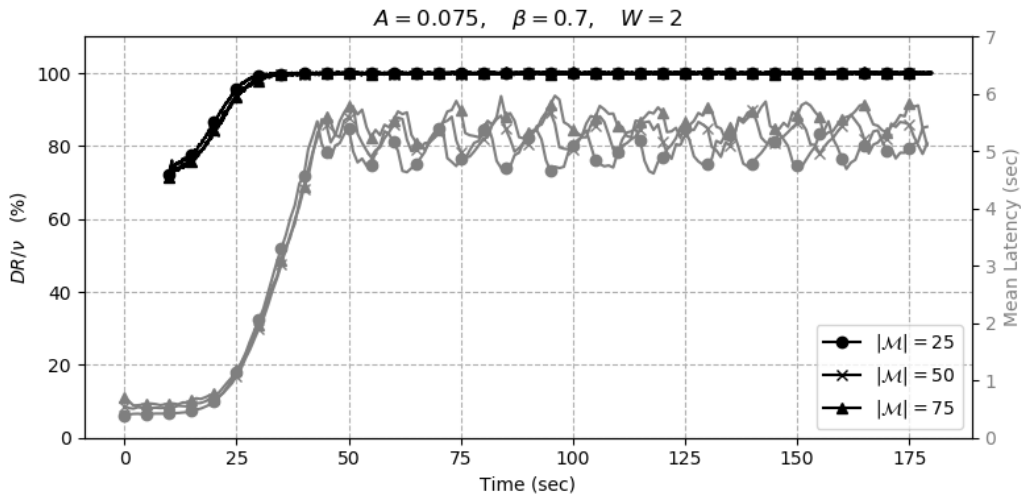


Figure 4.15: Combined dissemination rate as a percentage of ν, and mean dissemination latency with varying number of nodes $|\mathcal{M}|$ in the network.

low, blocks could be issued too rapidly for nodes with low writing power, and they would become overwhelmed.

For the purpose of illustration, suppose we estimate the combined computing power of all nodes in a network, and we set the PoW difficulty such that if all nodes were active at the same time, the nodes with the lowest writing power would just be able to keep up. We then have three cases that can arise:

1) some nodes are inactive, or the actual active computing power is lower than estimated;

2) the estimate of computing power matches the active computing power in the network;

3) the active computing power in the network is higher than estimated, or new nodes have recently joined with additional computing power, bringing the total above the estimated level.

Cases 1)–3) are compared to our algorithm in Figure 4.18. We simulate these cases using the network from Section 4.5.1, with computing power distribution as per the Zipf distribution of reputation given in this section. We set the PoW difficulty such that all nodes using their full computing power could issue blocks at a rate ν. For case 1), the inactive nodes as shown in Figure 4.2 are inactive, and all others use their full computing power, resulting in lower active computing power than estimated. For case 2), we take all nodes as active and utilising their full computing power, which results in the active computing power precisely matching the estimate. For case 3), we increase each node's computing power by 5%, resulting in higher active computing power than estimated.

From Figure 4.18 we see that: in case 1), latency is low but resources are underutilised as is evident from the dissemination rate below 80%; in case 2) the utilisation is good, but the latency gradually increases as some queues build up at the slowest nodes; in case 3) the utilisation is high but latency explodes due to nodes with low resources being unable to keep up with the rate of blocks being issued; with our algorithm we achieve high utilisation and maintain a stable latency as nodes can all manage their queues.

It is clear that PoW is very sensitive to the difficulty setting, performing poorly or failing completely if this is estimated incorrectly. This is a major issue because techniques for adapting the difficulty of PoW based on the estimated active computing power, such as that used in the Bitcoin network [2], operate over excessively long time scales of many hours or even days. In

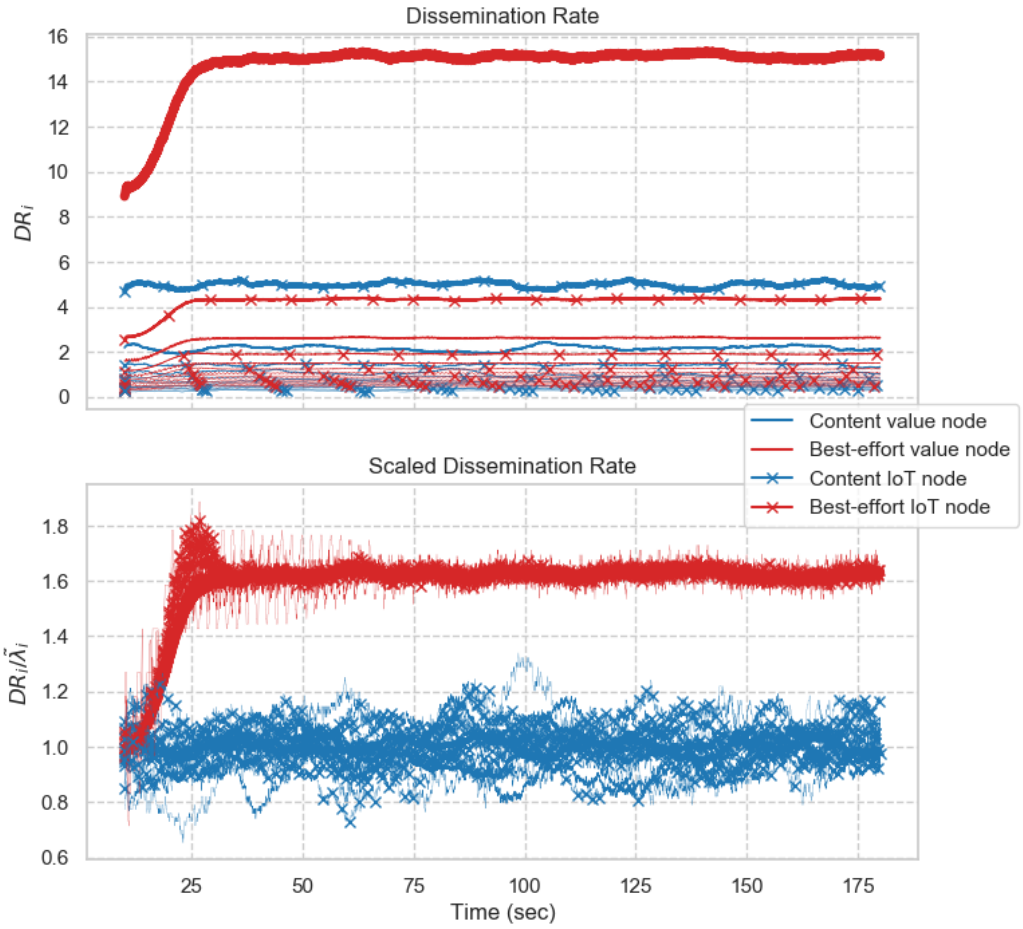


Figure 4.16: Dissemination rates of each node with a mixture IoT nodes and value nodes. The bottom plot demonstrates that fairness in dissemination rate is achieved in the presence of variable block work requirements.

our access control algorithm, on the other hand, nodes do not need to waste energy on solving PoW puzzles and can adapt their issuing rate immediately in response to traffic observed in their inboxes.

Remark: the results for PoW access control presented above demonstrate just some of the weaknesses of existing access control for DAG-based DLTs. Case 2) may seem almost acceptable, but note that this is not even practically achievable. Moreover, all three PoW cases correspond to hugely wasteful energy consumption and performance limitation, and the reality is that PoW is a legacy access control mechanism which prevents mainstream adoption of DLT in the IoT setting. This work offers the first IoT-friendly alternative.

4.6 Chapter Summary

We have presented an access control algorithm for DAG-based distributed ledgers which enforces a resource allocation to nodes based on their reputation. Our solution is especially suitable for the IoT setting because nodes are not required to wastefully commit computing resources in order to contribute to the ledger, as is the case in traditional PoW DLTs. Additionally, the DAG-based ledger structure permits high throughput and low latency because blocks are not limited to being added sequentially.

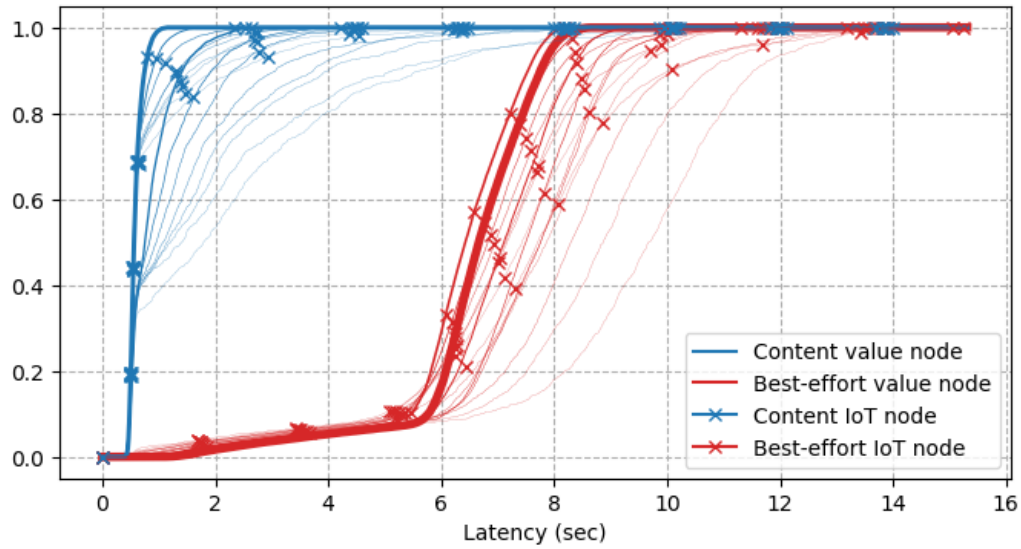


Figure 4.17: Cumulative distribution of dissemination latency for each node with a combination of IoT nodes and value nodes. It is clear that approximate fairness in dissemination latency is still achieved in the presence of variable block work requirements.

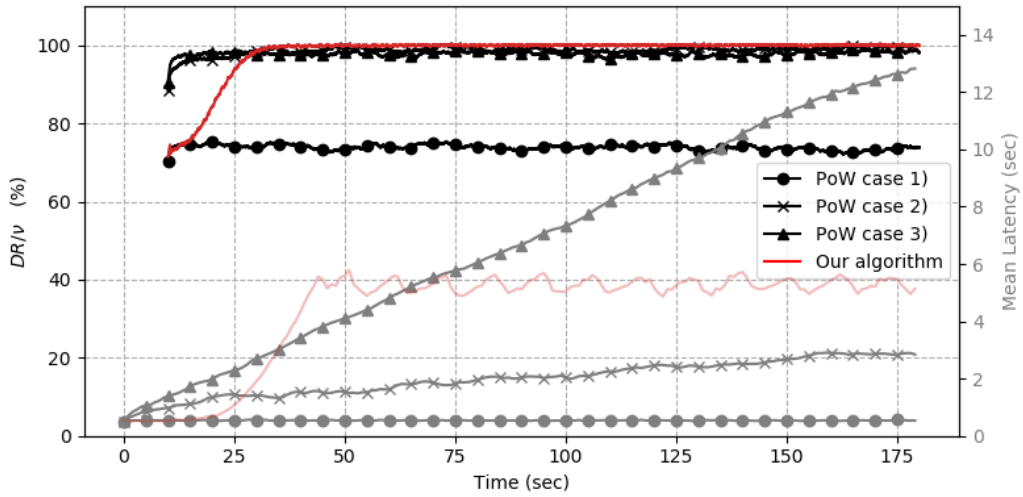


Figure 4.18: Dissemination rate as a percentage of maximum scheduling rate, ν, and mean latency for cases 1)–3) of PoW access control, shown alongside our algorithm with parameters given in Table 4.4.

The main features of our access control solution are an efficient fair scheduler, a TCP-inspired rate setting algorithm, and a buffer management scheme. We have shown, via network simulations, that our algorithm:

- ensures resources are allocated fairly and securely. The algorithm permits high utilisation of resources while preventing excessive congestion which could otherwise cause large delays and buffer overflows;
- is resilient against malicious agents wishing to claim more than their fair share of network resources;

- is robust to changes in parameters;
- permits a range of block types and sizes which may require differentiated treatment, making our algorithm applicable to networks with mixed node types, such as IoT nodes storing sensor data on the ledger, and larger devices making financial transactions;
- improves significantly on the state of the art in access control, namely PoW. Our comparisons demonstrate PoW's shortcomings and further motivates our new approach.

Chapter 5

Co-Design of Access Control, Tip Selection and Confirmation

***Abstract*— In this chapter, we present a co-design that unifies the work of all previous chapters, namely the tip selection analysis of Chapters 2 and 3 and the access control algorithm of Chapter 4, and we additionally confront confirmation in DAGs. This work not only allows us to understand each of the components of the data flow and how they interact more deeply, and to evaluate the end-to-end performance of the network in a more meaningful manner, but the preliminary results presented also demonstrate that excellent performance and security properties can be acheived. This is ongoing research in collaboration with a number of researchers from IOTA Foundation including Piotr Macek, Dr. Olivia Saa, Dr. William Sanders, Jonas Theis, Dr. Luigi Vigneri and Dr. Wolfgang Welz, as well as Lianna Zhao and Prof. Robert Shorten from Imperial College London. Other preliminary results related to this work and more extensive attack analysis also appear in [80].**

The core access control algorithm presented in Chapter 4 manages who gets to write what to a distributed ledger and, as such, the security of every part of that ledger relies on the access control component. Every decision made by nodes in a DLT (e.g., which transactions to confirm and which ones to reject) are based on the contents and structure of the ledger and metrics such as cumulative weight. For the sake of simplicity, we have mostly avoided discussion of the DAG structure in our discussion of access control up to this point, but the other key DLT components governing the DAG structure, such as tip selection, are inextricably linked to access control. In this chapter, we confront the link between access control and tip selection and present a new data flow which integrates tip selection and management of the DAG directly into the core access control framework.

Under normal operation of the core access control algorithm, each node schedules blocks at a maximum rate ν and allocates this scheduler capacity to the traffic of each node based on that node's reputation. This ensures a fair allocation of the network resources when demand is high. Additionally, nodes with blocks to issue regulate their issuing rate based on current traffic via the rate setter component. This ensures that the node's blocks are not delayed excessively or even dropped from the inbox of other nodes. If, however, a particular node does not follow the protocol and issues blocks at a greater rate than they should, some nodes may drop their blocks whilst others keep them and use them for tip selection. This can be problematic and prevent honest nodes' blocks from being scheduled if not carefully managed. Addressing these problems related to the DAG structure is the primary reason for the changes we propose here to the core access control algorithm of Chapter 4.

To develop these ideas further, recall that a node requires the entire *past cone* of any newly received block (all blocks it approves either directly or indirectly) to ensure that there are no

conflicts in what it approves and to compute quantities such as cumulative weight which may be required for consensus. We refer to a block without any missing information about its past cone as *solid*.

Definition 5.0.1 (Solid)**.** A block u is said to be solid for a node m if node m has seen all blocks in u's past cone, i.e., if node m has seen all blocks approved either directly or indirectly by u.

For the purpose of illustration of what it means to be solid, Figure 5.1 depicts the local ledger view of two nodes. The left node has not dropped the orange block, so the red block is said to be *solid*, while the node on the right has dropped the orange block, meaning that the red block is not solid for this node.

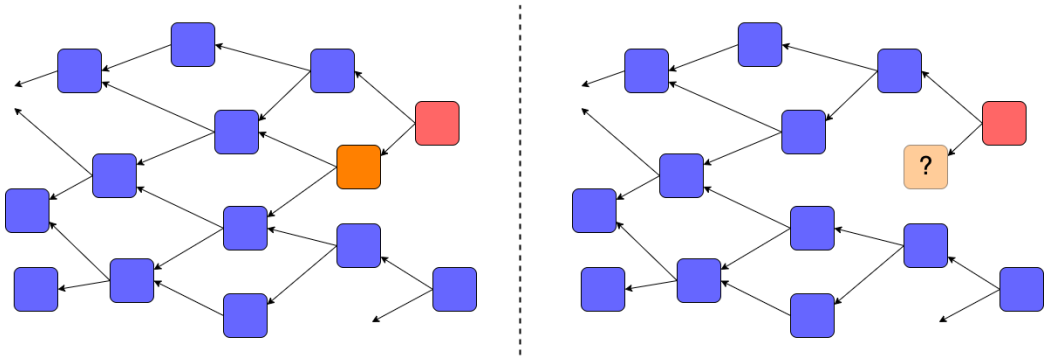


Figure 5.1: Left: the red block is solid for this node. Right: the red block is not solid for this node because the orange block was never received.

Blocks may be lost in transit from one node to another or arrive out of order for a variety of reasons. For example, a malicious actor could intentionally withhold certain blocks. However, the reason we are now concerned with solidification is that, as we observed in our experiments in Chapter 4, blocks from nodes which issue at excessively high rates may be dropped from scheduling buffers and hence may not be forwarded to neighbours. If blocks are dropped without consideration for their ordering in the DAG, this can result in blocks not being solid for neighbours. Nodes can request missing blocks from neighbours by sending them a *solidification request*[22], although we would like to avoid such requests as much as possible in the interest of minimising delays, backlogs and potential resulting cascades of dropped blocks. As we explain below, our new data flow minimises solidification requests by ensuring that blocks are always scheduled and forwarded in the order they appear in the DAG. Recall that communication between nodes in the peer-to-peer network is handled by TCP so packet ordering can generally be maintained effectively during data transmission.

Another key difference between the present chapter and Chapter 4 is in our evaluation. In Chapter 4, we evaluated our access control algorithm primarily based on dissemination rates (see Definition 4.3.2) achieved by each node and their dissemination latency (see Definition 4.3.3), which allowed us to show that nodes were allocated a fair throughput and delay based on their reputation. In this chapter, we are not only concerned with whether or not a block reaches all nodes (i.e., whether it is disseminated), but with how the DAG structure develops as a result and whether or not blocks become confirmed. In order to evaluate this, we require some new definitions.

Recall from Chapter 4 that $\mathcal{C}_m$ is the set of blocks confirmed by node m. We now define $\overline{\mathcal{C}}$ as the set of all blocks confirmed by *all* honest nodes, and $\overline{\mathcal{C}}^i$ is the subset of blocks in $\overline{\mathcal{C}}$ that were issued by node i. Furthermore, we define the notions of partial and full confirmation as follows.

Definition 5.0.2 (Partially Confirmed)**.** We say that a block is *partially confirmed* if the block is in $\mathcal{C}_m$ for some honest node m and it is **not** in $\overline{\mathcal{C}}$.

Definition 5.0.3 (Fully Confirmed)**.** We say that a block is *fully confirmed* if the block is in $\overline{\mathcal{C}}$.

[22] A special message requesting a missing block in order to make another block solid.

We then introduce the *confirmation rate* and *confirmation latency* metrics, defined as follows.

Definition 5.0.4 (Confirmation rate)**.** The *confirmation rate*, CR, is the rate of full confirmation of blocks, weighted by their work. This confirmation rate and the confirmation rate of node i's blocks, respectively, are defined as follows:

$$CR = \frac{\Delta W(\overline{\mathcal{C}})}{\Delta t} \tag{5.1}$$

$$CR_i = \frac{\Delta W(\overline{\mathcal{C}}^i)}{\Delta t} \tag{5.2}$$

where Δt is the time window over which we measure the confirmation rate and $W(A)$ denotes the work of blocks in the set A, as defined in (4.2).

Definition 5.0.5 (Confirmation Latency)**.** The confirmation latency of a block is the time from when the block is issued to when it is added to $\overline{\mathcal{C}}$. In other words, confirmation latency is the random variable of the time it takes for a block to become fully confirmed.

5.1 Model and Notations

The updated data flow, depicted in Figure 5.2, and modifications of the core access control algorithm aim to ensure that blocks are scheduled in the order they appear in the DAG, preventing solidification issues propagating through the network. By linking tip selection and tip set management explicitly to the scheduler, we aim to prevent honest nodes from selecting malicious tips and having their own blocks delayed as a result.

Table 5.1 provides notation relevant to the model presented in this chapter, with some notation carried over from Chapter 4 .

Table 5.1: Notation for node and network model.

$\mathcal{M}$	set of all nodes in the network
$\mathcal{N}_m$	set of nodes that are neighbours of node m
rep_m	reputation of node m
$\mathcal{C}_m$	set of blocks confirmed blocks by node m
CR	confirmation rate (all blocks)
CR_i	confirmation rate (blocks issued by node i)
ν	global block writing power
λ_m	issuing rate of node m
$\tilde{\lambda}_m$	assured issuing rate of node m

The actions outlined in Section 4.3, namely to receive, issue, schedule, write and forward are all still carried out by nodes in this modified data flow. However, the *writing* step is now dealt with explicitly by the *confirmation manager*, and the details of each of the processes are modified as discussed below. Our focus in this chapter is on the modified processes highlighted by the components of Figure 5.2.

5.1.1 Definition of Requirements

We now provide a new set of requirement for our modified access control algorithm which focus on confirmation of blocks rather than simply dissemination.

Definition 5.1.1 (Consistent Confirmation)**.** Consider a finite time window $w \in \mathbf{R}^+$, and a finite offset $h \in \mathbf{R}^+$. At time $t+h$, if all blocks added to $\mathcal{C}_m$ for any node m within time $[t-w, t]$ are in $\overline{\mathcal{C}}$ (see Definition 5.0.4), the access control algorithm of this network is said to satisfy the consistent confirmation requirement.

The interpretation of this consistent confirmation requirement is that if one node confirms a block, all nodes must eventually confirm this block, which is essential if consensus on the ledger is

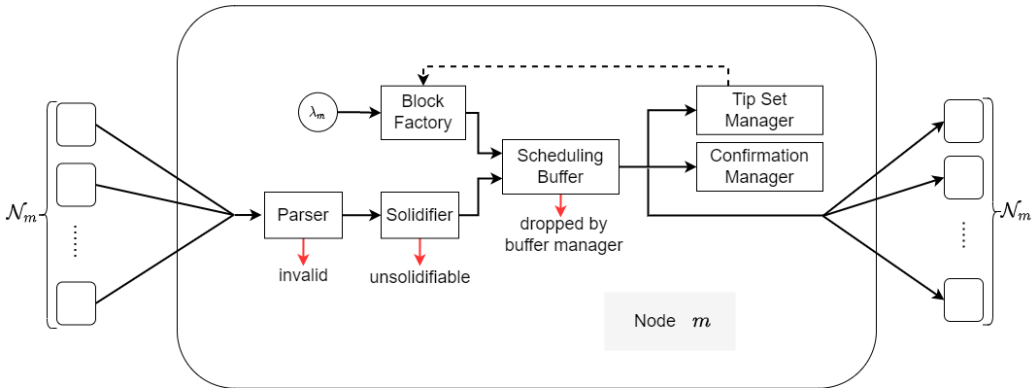


Figure 5.2: Node model: arrows indicate the flow of blocks through the node with red arrows indicating conditions under which blocks are dropped.

to be achieved. In other words, we will need to show that if a block becomes partially confirmed, it eventually becomes fully confirmed. The finite time window, w, after which we expect a partially confirmed block to become fully confirmed can be specified precisely in practice when evaluating whether a network achieves this property.

Definition 5.1.2 (Fairness in confirmation rate)**.** An access control algorithm satisfies the *fairness in confirmation rate* requirement if allocation of confirmation rate (see Definition 5.0.4) among nodes is max-min fair, weighted by each node's reputation. An allocation is max-min fair if an increase in any node's confirmation rate decreases the confirmation rate of another node m with equal or smaller *reputation-scaled* confirmation rate, CR_m/rep_m.

This fairness in confirmation rate requirement ensures that network resources are allocated to nodes based on their reputation.

Definition 5.1.3 (Fairness in confirmation latency)**.** We say that an access control algorithm satisfies the *fairness in confirmation latency* requirement if the expected confirmation latency (see Definition 5.0.5) of a node's blocks is independent of its reputation, and increases with its reputation-scaled confirmation rate.

This requirement ensures that blocks belonging to honest nodes do not experience excessive delays in being confirmed. This requirement also ensures that nodes attempting to achieve a higher confirmation rate than they should be entitled to will experience high delays.

Definition 5.1.4 (Security)**.** An access control algorithm satisfies the *security* requirement if the requirements defined in Definitions 5.1.1–5.1.3 are still satisfied in the presence of malicious actors.

This final requirement of security is essential for the public DLT environment in which some nodes (malicious actors) may try to gain an unfair advantage by deviating from the protocol. The security requirement ensures that this can not happen.

5.2 Access Control, Tip Selection and Confirmation

Our co-design spans the three main aspects of a DLT highlighted in Chapter 1, namely access control, tip selection and confirmation. As shown in Figure 5.2, the data flow is comprised of a number of interconnected modules which we now describe in further detail.

5.2.1 Rate Setter

The *rate setter* remains unchanged from that of Chapter 4, although rather than issuing blocks directly to the scheduling buffer, we now deal separately with block creation. As such, when a block is issued by the rate setter, it is passed to the block factory.

5.2.2 Block Factory

The *block factory* module is responsible for creating blocks to be issued to the ledger. This process includes selecting tips to which the new block is attached. Specifically, we use URTS (see Section 2.1.2) on a subset of tips which is determined by the tip set manager as described below. A timestamp is also added to every newly created block in the block factory.

5.2.3 Parser

Blocks received from neighbouring nodes go to the *parser* which checks basic validity information. For example, the parser verifies that the syntax of the block is consistent with any constraints of the DLT protocol and ensures that the timestamp of a block is greater than that of its parents. Invalid blocks are dropped and valid blocks are forwarded to the solidifier. In the simulations presented in this thesis, all blocks are valid and we assume that it is trivial to detect invalid blocks.

5.2.4 Solidifier

The *solidifier* module checks whether the full past cone of each new block has been received. If the parents of the new block are missing, the solidifier sends a solidification request to the node from which it received the new block. This may need to be done recursively if the parents of the requested parent are also missing, and so on.

5.2.5 Scheduler and Buffer Manager

The *scheduling buffer*, illustrated in Figure 5.2, consists of an inbox buffer, $Inbox_m$ (as described in Chapter 4), which is serviced according to the scheduler described in Algorithm 4 and managed according to the buffer manager described in Algorithm 5. Both of these algorithms are modified versions of those presented in Chapter 4. Note that there are only two modifications to the scheduler from the original access control algorithm: the first is that we order blocks by their timestamp rather than the time they become visible; and the second is that we require the blocks to be *ready* to be scheduled, which is defined as follows.

Definition 5.2.1 (Ready)**.** A block u is *ready* if u's parents are *scheduled*.

By scheduling in timestamp order and requiring blocks to be ready, we ensure that a node only forwards a block if it has also forwarded the block's parents and hence prevents solidification problems at neighbouring nodes. This also penalises nodes that attach their blocks to malicious blocks because their block will not be scheduled until the malicious parent is scheduled. When blocks are scheduled, they are passed to the tip set manager and confirmation manager and are forwarded.

Note that the timestamp of a block does not necessarily reflect standard time. However, it does provide an objective way to order the blocks because the timestamp is in the signed part of the block and can not be tampered with. Additionally, the ordering provided by these timestamps is consistent with the partial ordering provided by the DAG structure—this is enforced by the parser (see Section 5.2.3) which checks that a block's timestamp is newer than that of its parents.

The buffer manager also uses the timestamp of blocks to decide the order in which blocks are dropped. The consequence of a block being dropped is that it will not be used for tip selection or considered for confirmation and it will not be forwarded to any neighbours. However, dropped blocks are not removed from memory, so children of these blocks will still be solid and dropped blocks can be added back in to the scheduling buffer if they are needed in order to schedule a child block.

5.2.6 Tip Set Manager

The tip set manager provides the subset of tips $\mathcal{T}_m$ which should be used for tip selection by node m's block factory. To understand the rationale behind tip set management policy of Algorithm 6, note that the blocks of nodes that issue at an excessively high rate or otherwise misbehave will be delayed by the scheduler—we do not include these blocks for tip selection if they are delayed by

Algorithm 4 DRR−− Scheduler

Repeat for $i \in \mathcal{M}$ in a round robin cycle:
if $DC_m^i < DC_{\max}$ **then**
  $DC_m^i \leftarrow DC_m^i + Q_i$
end if
while Ready blocks in $|Inbox_m^i|$ **do**
  $u \leftarrow$ ready block in $Inbox_m^i$ with oldest timestamp
  if $DC_m^i \geq |u|$ **then**
    Schedule u
    $DC_m^i \leftarrow DC_m^i - |u|$
    Wait $\frac{|u|}{\nu}$ seconds
  else
    break
  end if
end while

Algorithm 5 Buffer Manager

if $|Inbox_m| > W_{\max}$ **then**
  $d \leftarrow \arg\max_{i \in \mathcal{M}} \frac{|Inbox_m^i|}{rep_i}$
  $u \leftarrow$ block in $Inbox_m^d$ with most recent timestamp
  Remove u from $Inbox_m$
end if

more than $h_{\max}$ seconds. Recall from Chapter 2 that the size of a node's tip set is proportional to the average delay between blocks being issued and reaching the tip set (denoted by h), so malicious nodes delaying their blocks can inflate the size of the tip set and reduce the chances of honest nodes having their blocks selected as tips and approved. Our data flow prevents this in two ways: old blocks (delayed in scheduler) are not included in the tip set; and blocks of severely misbehaving nodes are dropped by the scheduler and never make it in to the tip set manager in the first place. The formulae for tip set size derived in Chapter 2 can be used to approximately dimension the tip set capacity. For example, if we set $h_{\max}$ to be 3 seconds in a congested/fully utilised network with a scheduling rate $\nu = 250$ blocks per second, we can dimension the tip set for a maximum number of tips, $L_{\max}$, according to the worst case that all blocks are delayed by $h_{\max}$ using (2.29):

$$L_{\max} = 2\,\nu\,h = 1500$$

.

Algorithm 6 Tip Set Manager

Repeat each time a block u is scheduled:
$t \leftarrow$ current time
$t_u \leftarrow$ timestamp of u
if $t_u - t < h_{\max}$ **then**
  Append u to $\mathcal{T}_m$
  Remove any parents of u from $\mathcal{T}_m$
end if

5.2.7 Confirmation Manager

All scheduled blocks are passed to the confirmation manager where they are considered for confirmation. The tasks required by the confirmation manager depend on what notion of confirmation is used. As discussed in Chapter 2, there are two main types of confirmation that can be employed in DAG-based ledgers, namely weight-based and milestone-based, and we consider each of them in this chapter.

Algorithm 7 describes a weight-based confirmation manager. Each time a new block, u, is scheduled, the weight of this new block is added to the cumulative weight, $\mathcal{H}_u$, of the block itself

and to the cumulative weight of every block in its past cone. This cumulative weight update is performed recursively in Algorithm 7, and the cumulative weight of a block is not updated any further once it is confirmed. The only parameter required for weight-based confirmation is the cumulative weight threshold $\mathcal{H}_{\text{conf}}$ after which a block is deemed to be confirmed.

Algorithm 7 Weight-based Confirmation Manager

function UpdateCumulativeWeight(W, u, $\mathcal{H}_{\text{conf}}$)
    if status$_u$ = not updated **and** conf$_u$ = not confirmed **then**
        $\mathcal{H}_u \leftarrow \mathcal{H}_u + W$
        status$_u$ $\leftarrow$ updated
        if $\mathcal{H}_u \geq \mathcal{H}_{\text{conf}}$ **then**
            conf$_u$ $\leftarrow$ confirmed
        end if
        UpdateCumulativeWeight(W, v, $\mathcal{H}_{\text{conf}}$), for all v in parents of u
    end if
end function

Repeat each time a block u is scheduled:
conf$_u$ $\leftarrow$ not confirmed
$\mathcal{H}_u \leftarrow 0$
status$_v$ $\leftarrow$ not updated, for all v in scheduled blocks
UpdateCumulativeWeight($|u|$, u)

Algorithm 8 outlines a milestone-based confirmation manager. When a milestone block is scheduled, all blocks in its past cone are marked as confirmed. Once again, this recursive function terminates when it reaches a block that is already confirmed. Whilst no parameters are required for the milestone-based confirmation manager, one parameter that must be specificified for a milestone-based DLT network is the milestone period, T_{MS}, which specifies how often milestones are issued by the elected node or group of nodes responsible for issuing them.

Algorithm 8 Milestone-based Confirmation Manager

function AddMilestone(u)
    if conf$_u$ = not confirmed **then**
        conf$_u$ $\leftarrow$ confirmed
        AddMilestone(v), for all v in parents of u
    end if
end function

Milestone issuer
Repeat every T_{MS} seconds:
Issue a milestone block

All nodes
Repeat each time a block u is scheduled:
conf$_u$ $\leftarrow$ not confirmed
if u is a milestone **then**
    AddMilestone(u)
end if

5.3 Simulations

We now present simulations to verify that our new approach is effective. We have introduced components to our data flow which take the DAG structure into account, so we are now primarily concerned with whether each block is confirmed rather than simply disseminated. Specifically, we are now interested in verifying that the requirements laid out in Section 5.1.1 are satisfied. We begin by considering an honest environment in which all nodes follow the protocol, and we subsequently introduce a number of potential attack scenarios.

All simulations[23] in this chapter are carried out on a network of 20 nodes, unless otherwise specified, and connected in a random 4-regular graph topology where each communication link is bidirectional. 20 nodes is deemed sufficient to show the important properties of our system because it permits multiple nodes of each type and multihop paths between nodes while remaining computationally feasible to run the simulations. However, we shall also provide additional contrast experiments with larger networks to demonstrate that the results scale. As in Chapter 4, the mean propagation delay of each communication channel between neighbours is chosen uniformly at random between 50 ms and 150 ms, and the delay for each block on these channels is normally distributed around this average with standard deviation 20 ms. Node reputation is once again computed according to real data (the number of blocks issued by each account in the IOTA network) and follows a Zipf distribution with exponent 0.9. Simulation results are averaged over 20 Monte Carlo simulations. In this chapter, we increase the scheduling rate of nodes, ν, from 50 to 250. We also decrease the simulation time from 180 seconds to 60 seconds which is sufficient to demonstrate that a stable state is reached with this increased scheduling rate and we step the simulation in increments of 1 millisecond rather than 10 milliseconds. We assume that each node has buffer capacity greater than the parameter $W_{\max}$ specified for the buffer management, so no buffer overflows occur.

5.3.1 Honest Environment

The first sets of simulations we present is for an honest environment, i.e., a scenario in which all nodes follow the protocol and operate in one of the three honest modes of operation outlined in Definitions 4.3.4–4.3.6. The reputation distribution and modes of operation for each nodes is illustrated in Figure 5.3. Recall that we defined a confirmation manager from both weight-based confirmation (Algorithm 7) and milestone-based confirmation (Algorithm 8). The weight-based simulations provide clearer results than milestone-based simulations so we present only weight-based simulations here, and we show the milestone-based equivalents in Appendix D. The access control parameters were chosen experimentally for these simulations and are laid out in Table 5.2.

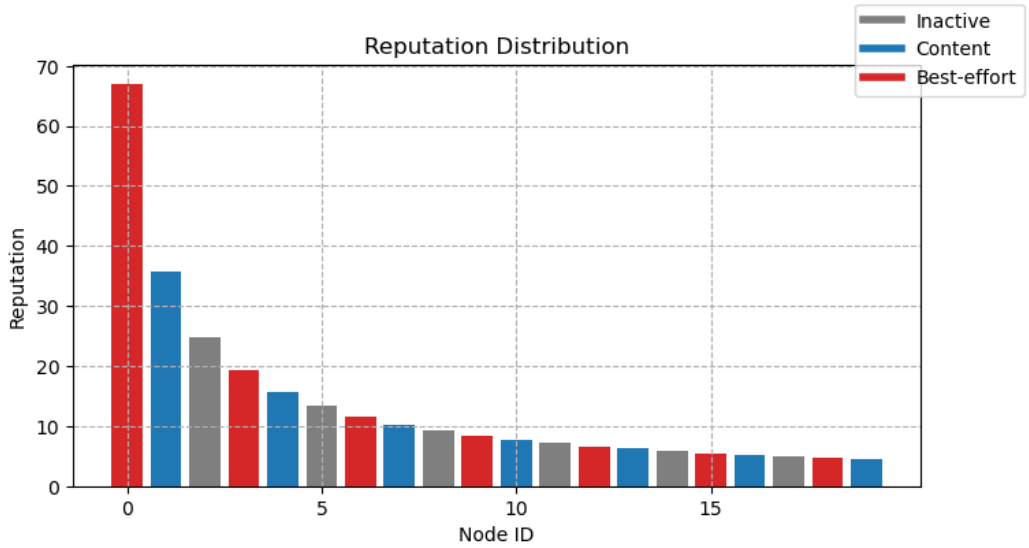


Figure 5.3: Reputation distribution follows a Zipf distribution with exponent 0.9. Nodes are content, best-effort, or inactive as indicated by each bar's colour.

Figure 5.4 shows the confirmation rate and mean confirmation latency over all blocks for this first set of simulations. The confirmation rate oscillates between approximately 200 and 300 blocks per second, with an average close to 250 blocks per second—recall that all blocks must pass through the scheduler to be confirmed so the scheduler rate of 250 blocks per second represents the maximum achievable average confirmation rate. The mean confirmation latency, on the other hand, settles

[23] Source code available at `https://github.com/cyberphysic4l/DLTCongestionControl/tree/thesis_ch5`.

Table 5.2: Access control algorithm parameters with weight-based confirmation.

Scheduler			**Rate Setter**				**Buffer Man.**	**Tip Set Man.**	**Conf. Man.**
ν	Q_i	$DC_{\max}$	A	β	τ	W	$W_{\max}$	$h_{\max}$	$\mathcal{H}_{\text{conf}}$
250	$\frac{rep_i}{\sum rep}$	1	0.075	0.7	0.2	1	500	3	200

at around 7 seconds. Note that if the scheduling rate of the network can be further increased, then confirmation latency can be further decreased.

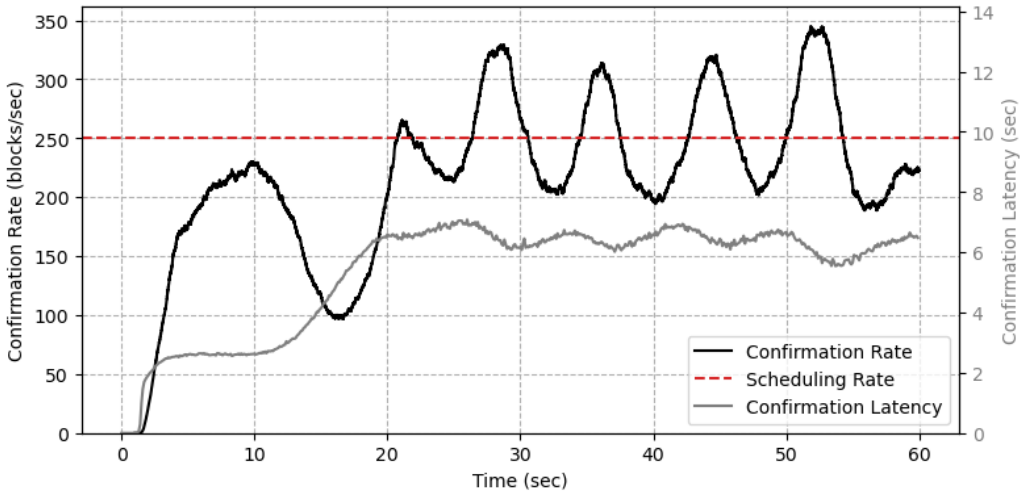


Figure 5.4: Confirmation rate and mean confirmation latency over all blocks.

Satisfaction of the consistent confirmation requirement is demonstrated by Figure 5.5 which shows the maximum time since partial confirmation (see Definition 5.0.2) for all partially confirmed blocks. The fact that this value reaches a steady state demonstrates that consistent confirmation is satisfied because it shows that all partially confirmed blocks eventually become fully confirmed.

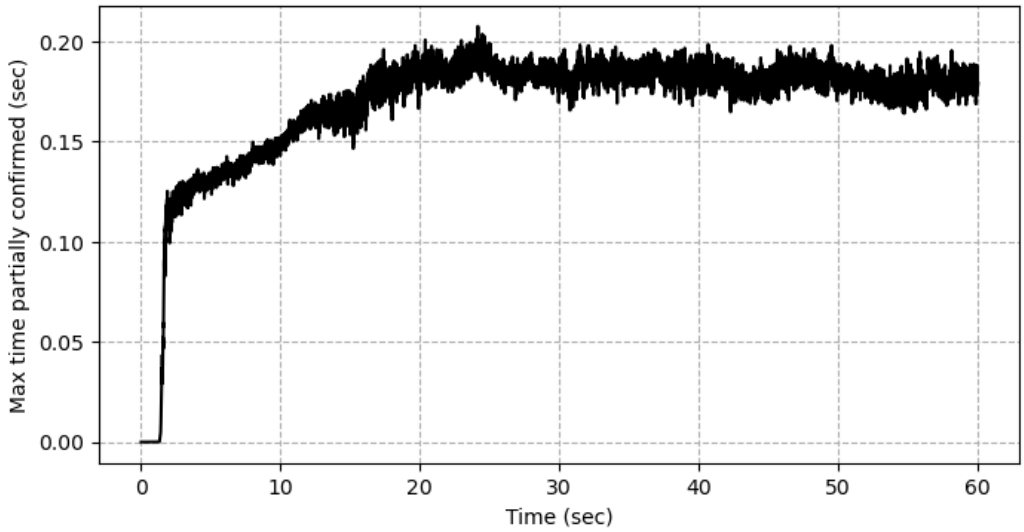


Figure 5.5: Maximum time since partial confirmation for all partially confirmed blocks.

Fairness in confirmation rate is demonstrated by Figure 5.6 which shows the confirmation rate CR_i for each node i in the network in the top plot, and these same confirmation rates scaled by

the reputation of the issuing node in the bottom plot. The bottom plot illustrates clearly that the confirmation rate achieved by each node is fair relative to their reputation.

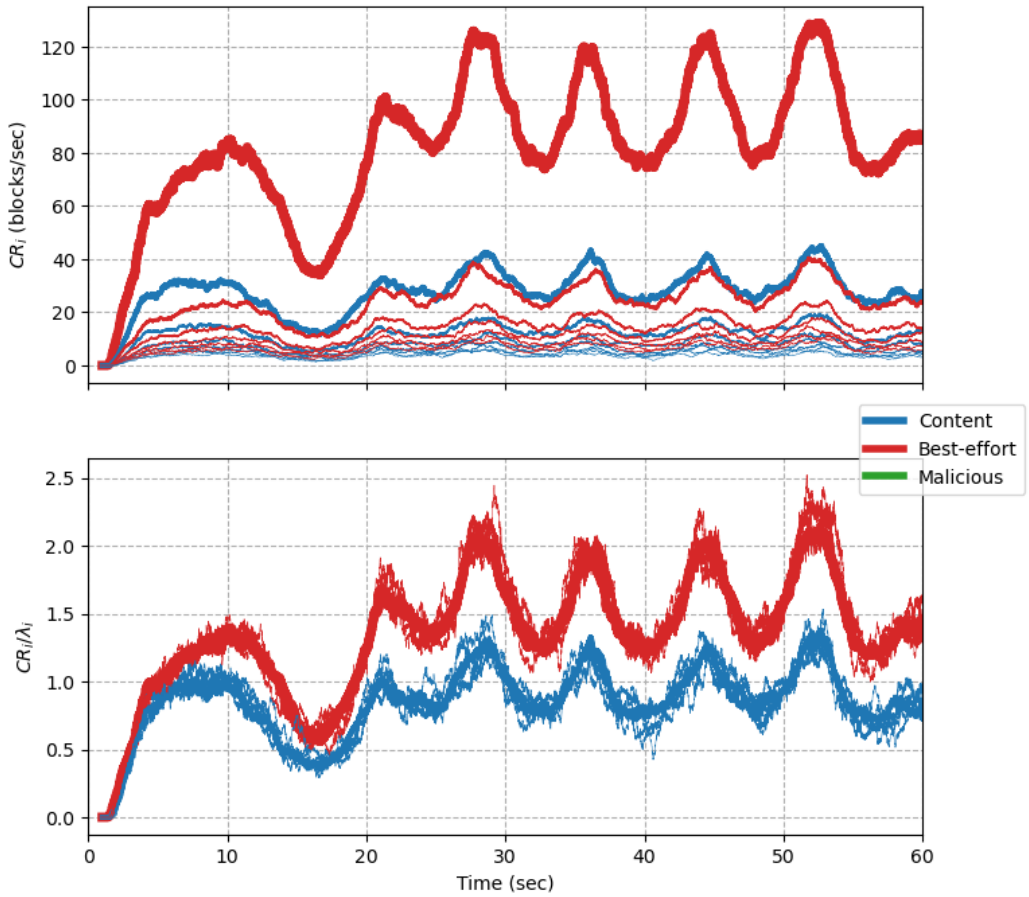


Figure 5.6: Confirmation rate (CR_i) and scaled confirmation rate (CR_i/λ_i) for each node i. The bottom plot scales each confirmation rate by their assured issuing rate, λ_i, which demonstrates that fairness in confirmation rate is achieved.

The fairness in confirmation latency requirement is verified in Figure 5.7 which shows the cumulative density of confirmation latencies for each node's blocks. It is clear that each node experiences approximately equal distribution of confirmation latency, relative to its reputation-scaled confirmation rate: best-effort nodes experience slightly higher confirmation latency than content nodes in line with their increased confirmation rate as shown in Figure 5.6.

In order to demonstrate how our approach scales, we next compare some results from networks with different numbers of nodes, $|\mathcal{M}|$. Figure 5.8 compares the confirmation rate and confirmation latency for networks of 20, 40 and 60 nodes. The only visibly significant difference between these results is that the confirmation latency is slightly increased as the network becomes larger. This can most likely be attributed to the increased expected number of hops required for a block to traverse a larger network which results in some delay in each block reaching all nodes.

Next, we compare the maximum time partially confirmed for the three networks mentioned above, as shown in Figure 5.9. It is clear that the consistent confirmation requirement is still satisfied in the larger networks. There is also an evident increase in the max time partially unconfirmed for larger networks which can once again be attributed to the increased number of hops for blocks to traverse the network.

Milestone-based simulations for this honest environment can be found in Appendix D.

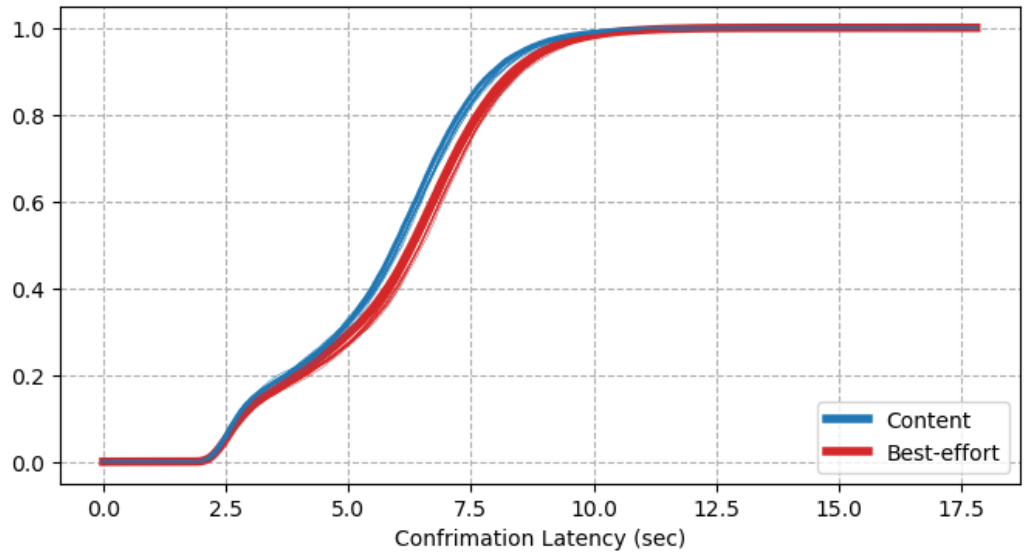


Figure 5.7: Cumulative distribution of confirmation latency for each node. This demonstrates that fairness in confirmation latency is achieved.

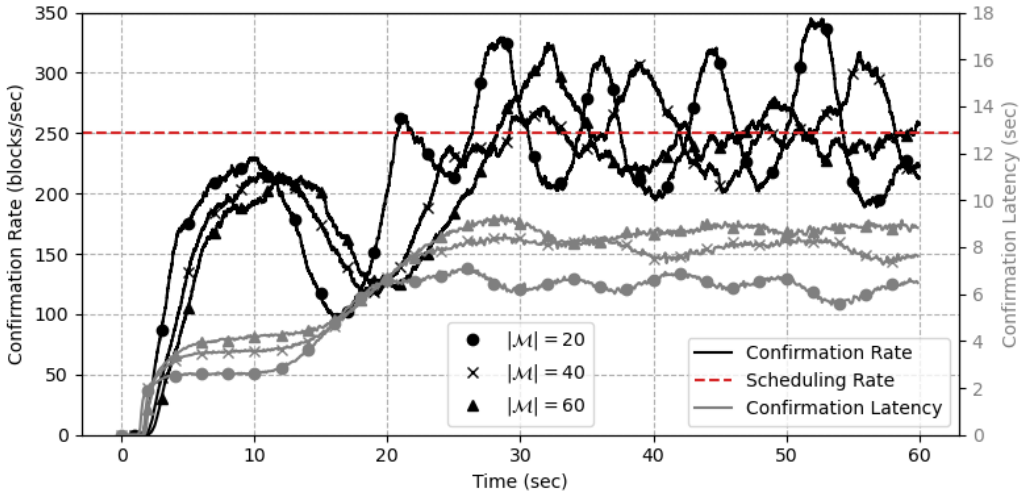


Figure 5.8: Confirmation rate and confirmation latency. Comparison of networks with 20, 40 and 60 nodes.

5.3.2 Adversarial Environment

In order to check the security requirement, we introduce malicious behaviour to our simulations. Specifically, we begin by introducing one malicious node, i, which issues blocks at 5 times their assured rate, $\tilde{\lambda}_i$ (see Table 4.1). The reputation distribution remains the same as depicted in Figure 5.3, but node 2 is now in malicious mode.

Figure 5.10 shows the confirmation rate and mean confirmation latency across all blocks for this set of simulations. The confirmation rate is now below 100% which is due to the fact that the malicious node's blocks get scheduled and hence take up some of the scheduler's capacity, but they do not get confirmed, as we shall see in other plots below. Note that although the confirmation rate is reduced, it can only be reduced as much as the reputation of an attacker allows because it is a result of the attacker simply wasting their allocated proportion of the scheduler throughput. The

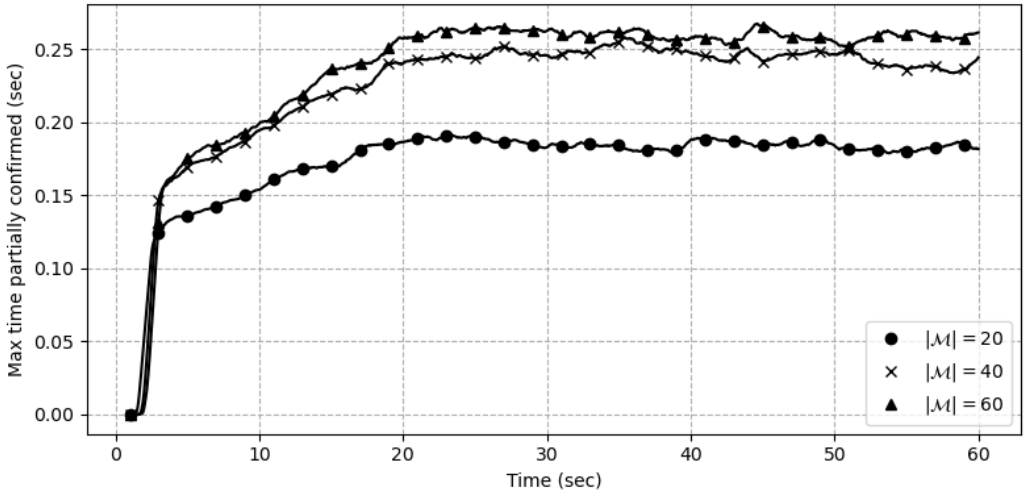


Figure 5.9: Maximum time since partial confirmation for all partially confirmed blocks. Comparison of networks with 20, 40 and 60 nodes.

confirmation latency is also increased compared to the honest environment due to the additional congestion introduced by the malicious node which was inactive in the honest environment.

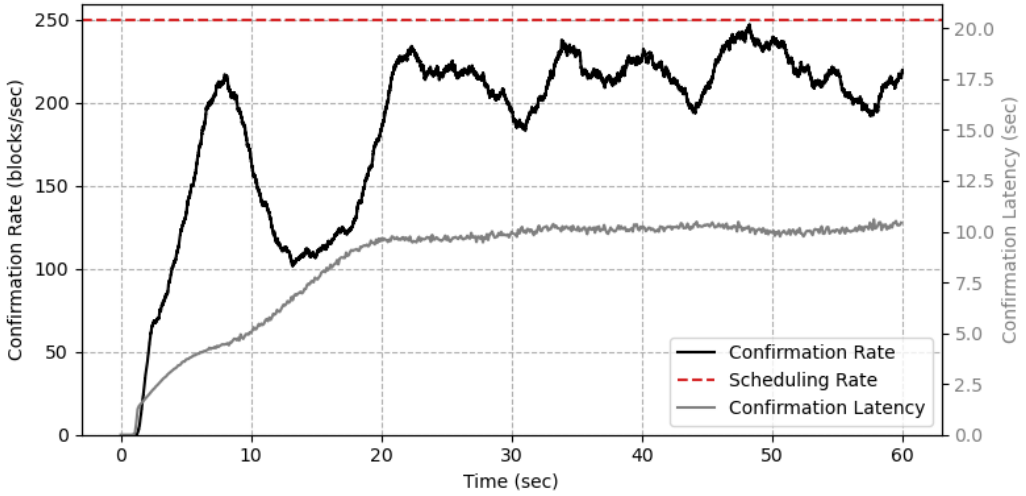


Figure 5.10: Confirmation rate and mean confirmation latency over all blocks.

The consistent confirmation requirement is not compromised by the malicious actor, as we can see from Figure 5.11, which shows the maximum time since partial confirmation across partially confirmed blocks. Despite the presence of a malicious node, this maximum time reaches a steady value which shows that all partially confirmed blocks eventually become fully confirmed.

Figure 5.12 demonstrates that the fairness in confirmation rate requirement is also still satisfied. As we can see, the honest nodes all achieve a fair reputation-scaled confirmation rate while the malicious node's confirmation rate drops to zero. The reason for the drop in the malicious node's confirmation rate is that the scheduler delays this node's blocks and, as a result, these blocks are not used for tip selection by any honest nodes so their cumulative weight does not grow. The fact that the malicious blocks are not used for tip selection also means that honest blocks are not affected as they essentially remain on a separate branch of the DAG whose cumulative weight grows as normal.

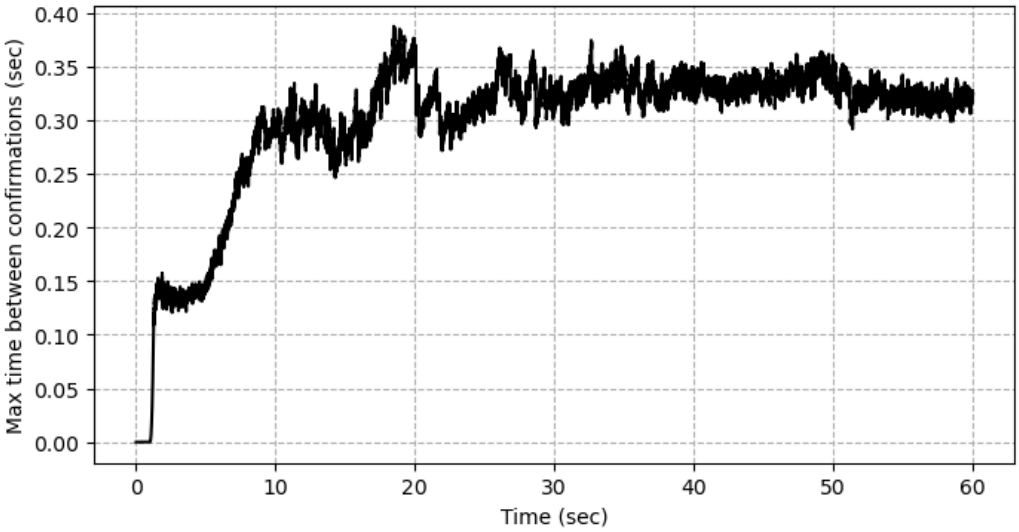


Figure 5.11: Maximum time since partial confirmation for all partially confirmed blocks.

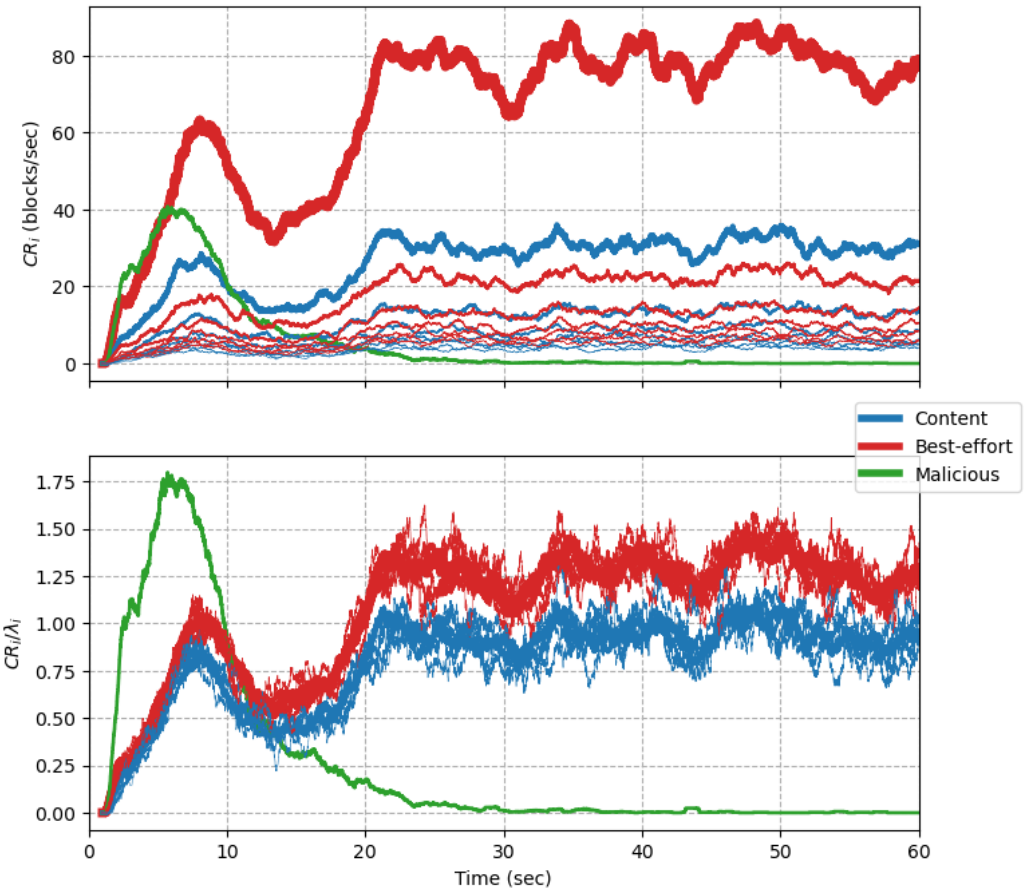


Figure 5.12: Confirmation rate (CR_i) and scaled confirmation rate (CR_i/λ_i) for each node i. The bottom plot scales each confirmation rate by their assured issuing rate, λ_i, which demonstrates that fairness in confirmation rate is achieved.

Figure 5.13 shows the confirmation latency of each node. Although it appears that the malicious node achieves a reduced confirmation latency, this is because the most of this node's confirmed blocks are early on in the simulation,before the other nodes have adapted their issuing rates to the network traffic conditions, and once the network has settled, the confirmation rate of the malicious node drops to zero. The honest nodes that continue to have blocks confirmed maintain fairness in their confirmation latency as in the honest environment.

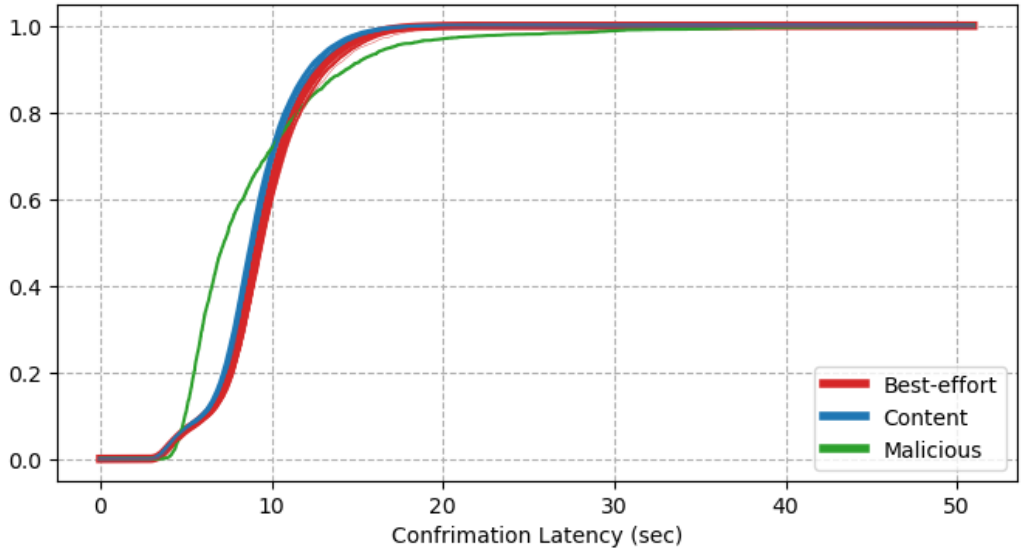


Figure 5.13: Cumulative distribution of confirmation latency for each node. This demonstrates that fairness in confirmation latency is achieved.

As before, milestone-based simulations for this adversarial environment are presented in Appendix D.

5.3.3 Tip Set Analysis

In the tip set analysis of Chapter 2, we used a simplified model in which we only considered a single node and we generated delays using a variety of probability distributions which sought to model real network processes. The simulator developed for this chapter provides a significantly more sophisticated model for a DLT network and allows us to study the tip set of each node individually.

First, we define tip set latency as follows.

Definition 5.3.1 (Tip set latency)**.** The tip set latency, H, of a block added to a node's tip set is the difference between the time the block is added to the tip set and the timestamp of the block (i.e., the time the block was created by the block factory).

Note that we referred to this quantity simply as a random delay in Chapter 2 because it was the only delay we were concerned with. The tip set latencies were also explicitly sampled from specific distributions of our choosing in Chapter 2 which sought to model the real delays experienced in DLT networks. We can now observe the actual tip set latencies and tip set sizes generated by our complete network simulator and compare the results to those of Chapter 2.

Beginning with results from the honest environment presented in Section 5.3.1, Figure 5.14 shows the CDF of the measured tip set latency, H, for the tip sets of each individual node in the network. We also include the following on the plot: the mean tip set latency, $h = 0.43$, highlighted by a vertical dotted line; an exponential CDF with rate $\mu = h^{-1}$; and a CDF for a uniform distribution with the same mean, h, lower latency limit $h_0 = 0.03$ and upper latency limit $h_1 = 0.83$.

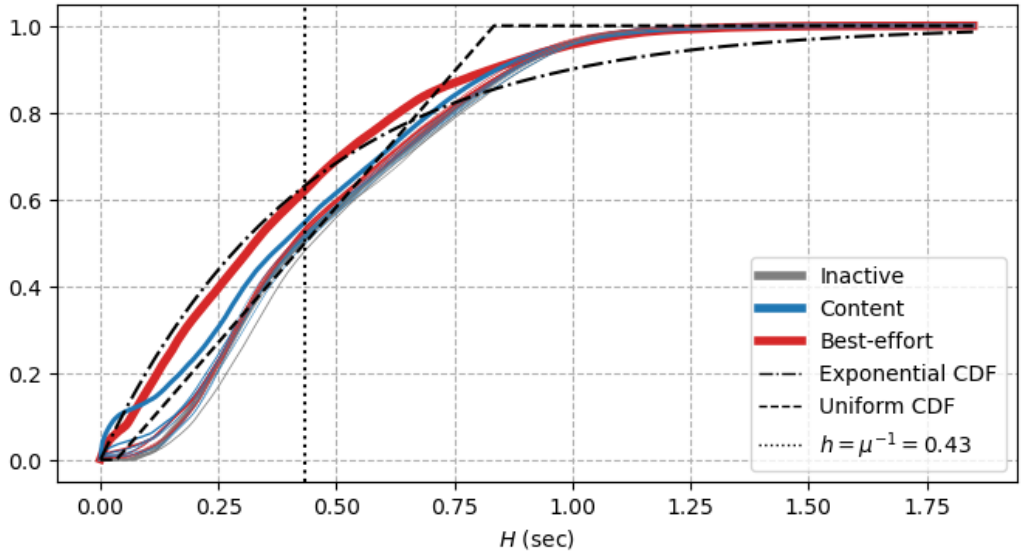


Figure 5.14: CDF of tip set latency, H, for the honest environment of Section 5.3.1. Also shown are the average latency, h, the exponential CDF with rate $\mu = h^{-1}$, and uniform CDF with $h_0 = 0.03$ and $h_1 = 0.83$ per Section 2.3.

From 2.32 we can compute the expected size of the tip set, L, according to the exponential distribution in Figure 5.14 as follows:

$$L = 1.2839\,\nu\,h = 1.2839\,(250)\,(0.43) \approx 138 \tag{5.3}$$

In the case of the uniform distribution plotted in Figure 5.14, the expected size of the tip set can be obtained from (2.34) with $h_0 = 0.03$ and $h_1 = 0.83$ as follows:

$$L = 1.72\,\nu\,h = 1.72\,(250)\,(0.43) \approx 185 \tag{5.4}$$

With these predictions of tip set size in mind, we can plot the actual tip set size as measure in our simulations. Figure 5.15 shows the number of tips in each node's tip set in the honest environment simulations. We can see the tip set size for each node converges to an average value of around 168 tips, which lies between the two predicted values associated with the exponential and uniform distributions.

Next, we consider the equivalent results for the adversarial environment of Section 5.3.2. Figure 5.16 shows the CDF of H as measured at each node. We can see that around 45% of the blocks added to the malicious node's tip set experience zero latency—these are the malicious node's own blocks which bypass the scheduler and are immediately added to the tip set by the malicious node. The CDFs of honest nodes maintain a similar shape, but the average tip set latency, h, is increased. However, many of the malicious node's blocks do not even make it in to the tip set as they have been delayed by more than $h_{\max}$ (3 seconds in these simulations).

From 2.32 we can compute the expected size of the tip set, L, according to the exponential distribution in Figure 5.16 as follows:

$$L = 1.2839\,\nu\,h = 1.2839\,(250)\,(0.63) \approx 202. \tag{5.5}$$

In the case of the uniform distribution plotted in Figure 5.16, the expected size of the tip set can be obtained from (2.34) with $h_0 = 0.13$ and $h_1 = 1.13$ as follows:

$$L = 1.80\,\nu\,h = 1.80\,(250)\,(0.63) \approx 283 \tag{5.6}$$

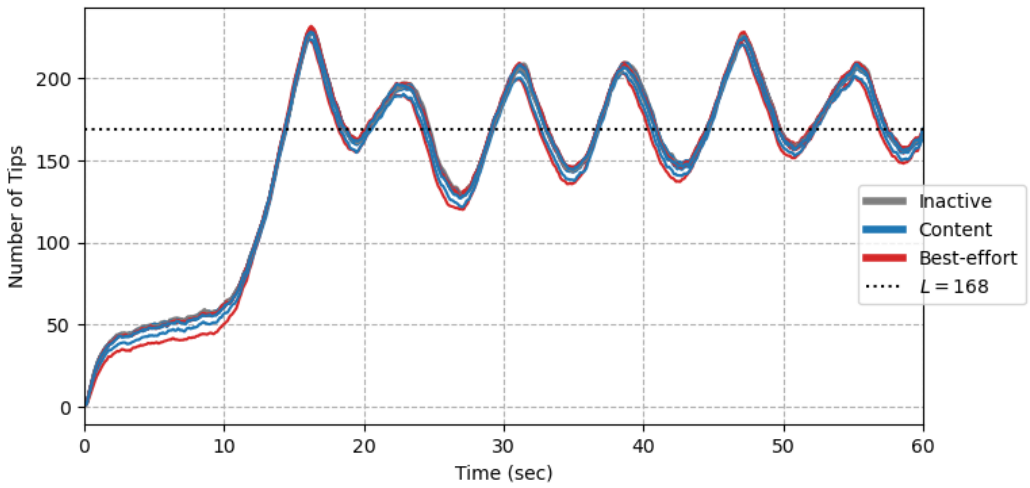


Figure 5.15: Tip set size for each node in the honest environment of Section 5.3.1. The dotted line highlights the average tip set size over the final 40 seconds of the simulation when it has stabilised.

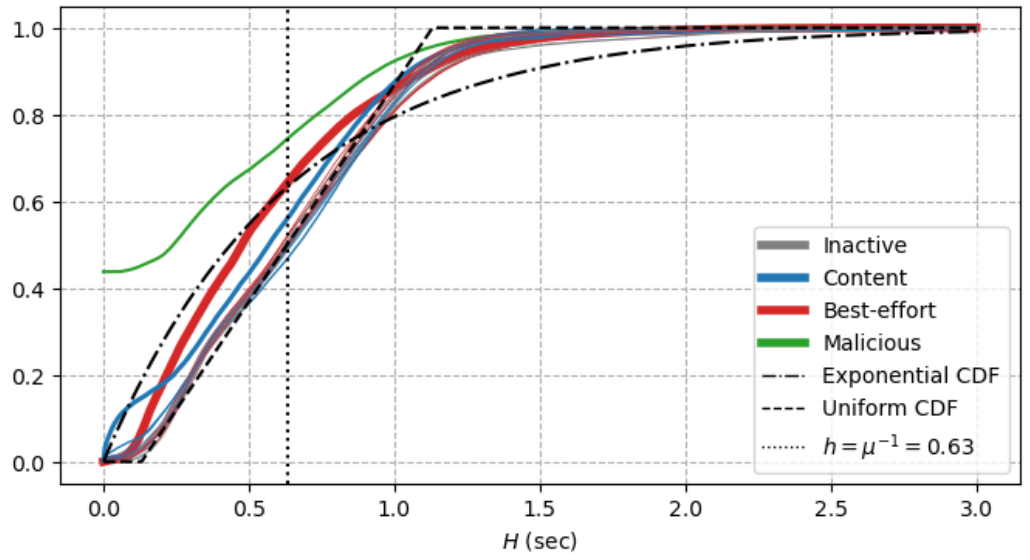


Figure 5.16: CDF of tip set latency, H, for the adversarial environment of Section 5.3.2. Also shown are the average latency, h, the exponential CDF with rate $\mu = h^{-1}$, and uniform CDF with $h_0 = 0.13$ and $h_1 = 1.13$ per Section 2.3.

We can then once again compare these predictions to the actual tip set sizes measured in our simulations, which are shown in Figure 5.17. The average tip set size of the honest nodes, $L = 214$, most closely matches the value of 202 predicted by the exponential distribution. The tip set of the malicious node, on the other hand, begins at a similar size to those of the honest nodes, but decreases as the simulation goes on because honest nodes stop selecting the malicious node's blocks as tips due to their tip set latency exceeding $h_{\max}$. When the honest nodes stop selecting the malicious blocks as tips, they no longer get selected multiple times due to delays which is the process that results in a increases in the size of a tip set.

Figure 5.18 shows the tip set size for each node over a single simulation of 120 seconds in order to show that the malicious node's tip set size decreases drastically, but the honest nodes' tip sets are not affected.

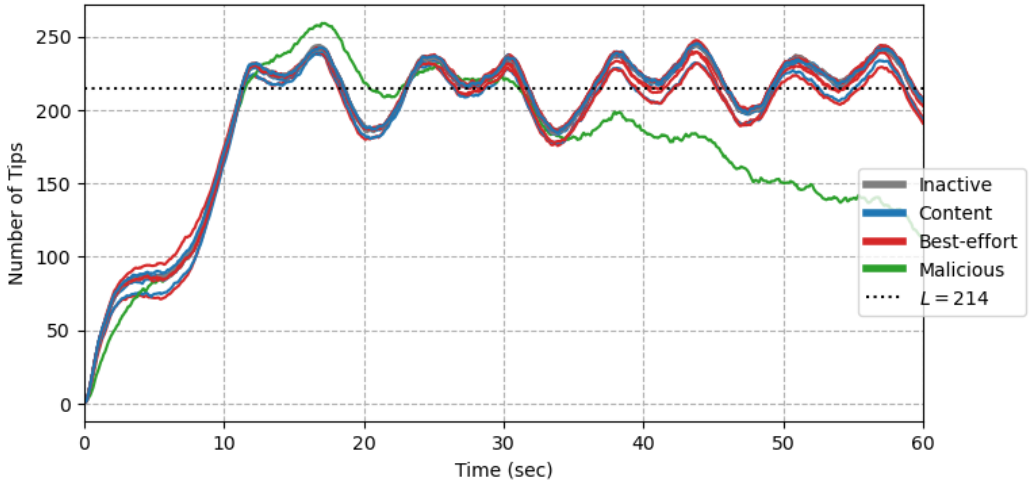


Figure 5.17: Tip set size for each node in the adversarial environment of Section 5.3.2. The dotted line highlights the average tip set size of the honest nodes over the final 40 seconds of the simulation when it has stabilised.

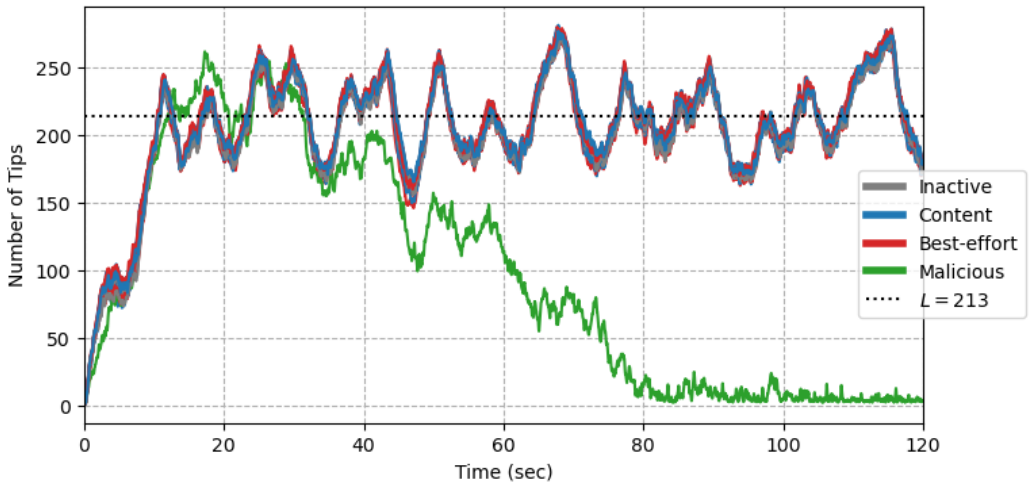


Figure 5.18: Tip set size for each node in the adversarial environment of Section 5.3.2. The dotted line highlights the average tip set size of the honest nodes over the final 100 seconds of the simulation when it has stabilised.

5.4 Chapter Summary

We have presented an extension of the access control algorithm in Chapter 4 which additionally accounts for both tip selection and confirmation processes. The preliminary results presented focus on confirmation rates rather than simply dissemination rates which provides a more meaningful evaluation of the effectiveness of our approach. Results for weight-based confirmation have been included above, and equivalent results with milestone-based confirmation are included in Appendix D. We have also presented analysis of how the tip set of each node in our simulations evolves and compared this with previous results from a more primitive simulator in Chapter 2.

The co-design presented here is the subject of intense ongoing research with IOTA Foundation including implementation and demos in the GoShimmer network [56]. Future work of immediate importance will include extended analysis of attacks, including attacks in which malicious nodes make use of alternative tip selection strategies.

Chapter 6

User-Node Interaction Mechanisms for DLTs in Enterprise Applications

***Abstract*— Despite growing interest in DLT across a wide array of industries and continued academic research in the area, distributed ledgers are not yet ubiquitous, even in highly technologically advanced sectors. One explanation for this delay in their adoption for more practical applications is that little research has focussed on usability of DLTs. In this chapter, we propose a mechanism for users to interact with DLT networks without needing to operate as a fully-functional node. The focus of our solution is on quality of service with a view to bridging the gap between everyday users of these technologies who simply wish to make use of the ledger, and the node operators who support this service and invest thier resources into the functioning of the network. The work in this chapter is the first to focus on network usability for DAG-based DLTs and aims to pave the way for further research in this direction. This is joint work with Dr. Luigi Vigneri of IOTA Foundation and Lianna Zhao and Prof. Robert Shorten of Imperial College London.**

Distributed ledgers have been intensely studied in recent years, and the main topics of these studies concern aspects of their design such as consensus mechanisms, scalability and security, some of which have been addressed in the preceeding chapters of this thesis. A number of real use cases and applications have also been studied, for instance in the areas of transportation [81, 82]. In particular, ledgers based on *directed acyclic graphs* (DAGs), have arisen as a promising solution for IoT applications because they can facilitate high throughput and low delays, and they present fewer barriers to participation than blockchain alternatives [26]. As DLTs and their applications become better known and accepted in the wider technology community, the related issue of *network usability* is being recognised as a bottleneck issue hindering applications of this nascent technology. In some sense, this is to be expected as the underlying technology is not yet mature. Nevertheless, the fact that in many ledger architectures, users report lengthy and variable times for transactions to complete, as well as unfairness in the service experienced by users of the technology, is a significant impediment to its adoption. Consequently, the issue of network usability represents a significant research challenge for the DLT community.

In many DLT architectures, the variation in the experience amongst users can be attributed to two factors. The first comes down to the design of core components of the ledger architecture and how these affect the efficiency and performance of the system. The second, equally important factor, is connected to the interaction between human decision makers and the DLT network, and the incentive structure that guides this interaction. This second factor is our primary concern in this work. In a typical DLT architecture, two distinct types of network actors can be readily identified. First, there are individuals or organisations who maintain a copy of all transactions on the ledger and participate fully in network activities such as consensus and validation. These actors, which we refer to as *nodes*, have been the primary focus of all research presented up to

this point of the thesis. They have the power to modify the ledger to include transactions which they may use to transact themselves or may offer this as a service, perhaps for economic gain. A second type of network actors are basic actors, referred hitherto as *users*, who are simply interested in utilising the network as part of some enterprise. For users (humans or software agents), the network is a tool to be utilised as part of some application, and their only objective is to send transactions through the network for a specific purpose. Users can only add data to the ledger by sending it to a node, usually via a digital wallet, and in many cases these users are willing to pay a fee to nodes to achieve a good *quality of service* (QoS).

The experience of users is strongly influenced by the manner in which *users* and *nodes* interact, which is driven by the different factors that motivate each actor's participation in a DLT network. Typically, *users* wish to transact as cheaply as possible while receiving some level of QoS from the network. For example, *users* may wish to transact with guaranteed delay; minimise the financial cost of transactions or their energy consumption; or be simply assured of fairness and value for money they receive with respect to other *users* of the network. *Nodes*, on the other hand, are driven by the desire to use of their resources (e.g., computational power or reputation[24]) for both personal or societal gain. The desires and objectives of *users* and *nodes* are not always mutually inclusive. In other infrastructures, the interaction between users and nodes can sometimes lead to unstable network behaviour. For example, parallels can be found in road networks, where *toll roads* (nodes) are attractive to vehicles (users) precisely because they offer faster and more reliable transit times. However, if they attract too much traffic, precisely the opposite can be the effect, leading to unpredictable and sometimes catastrophic user experience. Problems of this nature are not unique to transportation and arise also in other domains - for example in job scheduling and queuing problems that arise in many applications (we shall have more to say on this in the next section). Here we simply note that similar issues can arise in DLT networks if the interaction between users and nodes is poorly designed [83] and while node-user interaction mechanisms have been designed for blockchains (based on transaction fee incentives), there is currently no user-node interaction mechanism designed for DAG-based DLTs of the kind considered here. The problem of designing such a mechanism for DAG-based ledgers differs fundamentally from that of blockchains because nodes can add transactions to DAGs in parallel rather than sequentially as in blockchains. This feature of DAGs restricts users of these networks to selecting specific nodes to process each of their transactions and results in an entirely new paradigm for user-node interaction.

Our goal in this chapter is to suggest an interaction mechanism between nodes and users which allows both users and nodes to achieve their objectives and leads to networks that are stable, robust and fully utilised. Our idea is simple. Nodes broadcast a *QoS indicator* for the service they can offer, and users respond probabilistically to this signal. We shall show that this simple algorithm equalises the QoS experienced by users, avoids large deviations in QoS experienced by individual users, whilst allowing nodes to be rewarded for the service they provide to the network. Moreover, the mechanism is inherently robust to the behaviour of dishonest nodes and users. Nodes advertising false QoS information can be rapidly detected and blacklisted by users. The proposed policies are also agnostic with respect to specific implementation issues, such as node discovery, and assume users have full access to QoS signals from nodes. As a final comment on the contribution of this work, it is important to note once again that the DAG-based DLT setting considered here requires a fundamentally different user-node interaction mechanism than those found in traditional blockchain architectures such as Bitcoin. The key difference lies in the fact that users must select a specific node to issue their transactions in our setting rather than broadcasting them to as many nodes as possible. To the best of our knowledge, this work is the first to consider the design of a user-node interaction mechanism for DLTs of this kind.

This chapter is structured as follows. In Section 6.1, we present related work. In Section 6.2, we give a basic system model for the relevant components of DAG-based DLT networks including some basic concepts, such as users, nodes and QoS indicators. In Section 6.3, we present a user-node interaction mechanism for DAG-based DLT networks and propose a variety of policies for users and nodes to orchestrate their relationship. The proposed policies range from naive approaches to more sophisticated strategies which take into account different indicators of QoS. In Section 6.4, a set of simulations are performed to validate and contrast the proposed policies.

[24]Reputation is a numeric value associated to a node which affords it ledger access. In principle, reputation should be difficult to gain and easy to lose.

6.1 Related Research

Before proceeding, it is worth noting that the research in this chapter builds on a number of seemingly unrelated prior works. First, the issue of user-network interaction arises in several domains. For example, stochastic policies to guide this interaction (of the nature proposed here) have been studied and analysed in the context of transportation networks [84, 85, 86, 87]. The authors in [88] and further work in [89] propose an algorithm enabling the number of arriving, departing cars and the instantaneous occupancy to be counted by car parks. Based on broadcasts from the car parks, cars then can predict the parking space availability at the estimated time that the car will arrive there. In [90], the authors propose a stochastic policy to associate cars with parking spaces and balance cars across a network of car parks and charge points in a manner that minimises the probability of a space not being available when cars arrive at a car park. The problem discussed in this chapter is also closely related to a host of other load balancing problems that can be found in the networking literature—one example is that of server farms with immediate dispatching of jobs requiring task assignment policies [91]. Well-known policy examples that arise in this context include the *random selection policy*, *the round-robin selection policy* and the *join the shortest queue policy* (the algorithms proposed in this chapter are variants of the random selection policy in which a non-uniform distribution is sampled to select a node). Examples of relevant work in this direction can be found in [92, 93, 94, 95, 96, 97, 98] and the references therein.

It is also worth noting that the user-node interaction problem arises in the design of other DLTs, namely those based on blockchain technology. The best known such network is the Bitcoin network. In Bitcoin each miner (node) maintains a pool of transactions referred to as a *mempool* [91]. Miners select transactions from their mempool to include in block proposals, and if they are successful in adding a block, they receive fees from all the included transactions. In the blockchain setting, users aim to send their transactions to the mempools of all miners to maximise their chances of having them included in a block. Due to the sequential nature of how blocks are appended to a blockchain, miners can share identical mempools without the risk of adding the same transaction to the ledger twice. As such, users do not need to send their transactions to specific nodes in blockchain networks but should aim to get their transactions to as many nodes as possible. In the DAG-based DLT setting we consider here, however, nodes cannot share mempools because transactions can be added in parallel rather than sequentially and nodes would risk adding the same transactions more than once causing conflicts in the ledger. While the ability to add transactions in parallel is a desirable property for a DLTs, particularly in IoT settings, it renders the design of a user-node interaction mechanism significantly more complex. Some primitive techniques to improve the usability of DAG-based DLTs have been employed in practice, for example, some of the more strenuous tasks required to issue a transaction were offered as a service by a third party provider in a early implementation of the IOTA network[25]. However, to the best of our knowledge, no solutions for improving the interactions between users and nodes have been proposed in this setting.

6.2 System Model

We consider the network architecture as depicted in Figure 6.1 which shows users accessing a DLT network via nodes, and nodes which communicate directly with one another, forming a distributed ledger network. More specifically, users make a selection from a set of nodes according to some criteria, and then send their transactions to these nodes to be processed. Furthermore, in this initial simplified model (neglecting some networking details such as availability of IP addresses) where users are able to query all nodes, users are free to select a node to process their transactions, and nodes are free to accept or reject transactions from individual users. It is important to note that, in contrast to blockchain-based DLT networks, users only send each transaction to a single node, and nodes do not share these transactions with one another prior to issuing them to the ledger.

Nodes may offer ledger access as a public service and can offer different levels of QoS to users (some nodes may be operated by private organisations who reserve resources for their own gain, however, we do not consider such nodes any further in this work). Generally speaking, nodes

[25] https://ecosystem.iota.org/projects/powsrv-io

must consume resources in order to add transactions to the ledger, and providing better QoS consumes more resources. For example, in a PoW ledger, the consumed resource is computation power, whereas in PoS ledgers, the consumed resource is wealth in the native ledger currency. A generalised version of PoS known as delegated PoS allows this resource to be transferred to nominated nodes, serving as a proxy for reputation. In the remainder of this chapter, we adopt the PoR model and we refer to this resource simply as reputation, where the reputation of a node i is denoted rep_i, as in earlier chapters. Although small inconsistencies in reputation calculation across nodes can typically be permitted and reputation can be changed over time, we assume that each node's reputation is a constant quantity that is agreed upon and publicly known. In what follows, we also make the following assumptions.

- Ledger access is limited by scarce resources so providing ledger access with low delay is costly to nodes. This is generally true for all DLT architectures (beyond the DAG-based DLTs considered here), for example, in a conventional PoW-based ledger, a node must consume more power to find PoW solutions more frequently and issue transactions with lower delay.

- Each node in our network is equipped with a Local Transaction Pool (LTP) that is used as a buffer to store pending transactions. Transactions sent from a user to a particular node enter that node's LTP. Note that similar pools of transactions can also be found in other ledger architectures, such as the mempool in the Bitcoin network, although nodes typically share a common mempool which is not permitted in our setting.

- Nodes can advertise their expected QoS that they can offer via some proxy. For example, nodes may advertise the expected delay from receiving a new transaction to writing it to the ledger. Additionally, nodes may require a fee for issuing a transaction and this can also be taken into account as part of a node's overall QoS.

A model for our user-node interaction mechanism is depicted in Figure 6.1. The sending rate of user i is denoted by μ_i, while the service rate of node i's LTP is denoted by λ_i. The QoS indicator for each node is represented by the coloured bars above their LTP. Here, a red QoS indicator simply means poor QoS is offered by this node (for example, this could be due to high delay and/or high fees), while yellow indicates average QoS and green indicates good QoS.

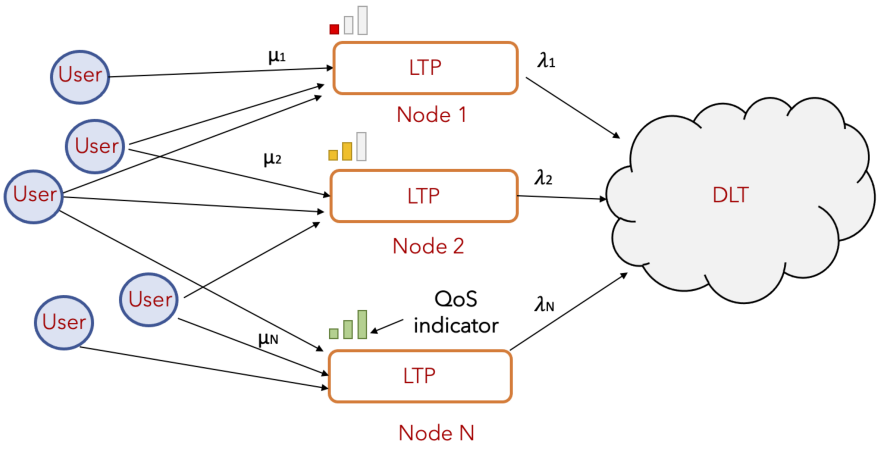


Figure 6.1: Basic network model for user-node interaction mechanism.

In what follows, we are specifically interested in designing a mechanism to orchestrate the interaction between users and nodes in DAG-based DLT networks. Our goal is to deliver a uniformly good QoS to all users and ensure that the probability of any user experiencing a very bad QoS is low, while allowing nodes to profit by processing a regular stream and (on average) fair share of transactions. While our results generally apply to any DAG-based DLT, we evaluate our solution using the DLT architecture described in Chapter 4 which includes PoR access control. In other words, each node regulates its own issue rate via a distributed algorithm based on the additive

increase multiplicative decrease (AIMD) algorithm [99, 100, 101]. Fair issue rates are then enforced throughout the network via a scheduling algorithm based on deficit round robin (DRR) algorithm [69]. The result of this mechanism is a network that is attack resistant and where each node achieves a transaction issue rate proportional to its *reputation*.

While the definition of QoS is quite broad, we will focus our attention on delay and fees as measures of QoS in this work. In order to provide an indicator of QoS to users, nodes must estimate the expected delay for them to issue a newly arriving transaction. This delay depends on two factors: the number of transactions in the node's LTP at the time of the new transaction's arrival; and the service rate of the LTP. As mentioned above, in the IOTA network, the service rate of each node's LTP is determined by the AIMD-based rate setting algorithm employed by the node. Given this background, we now propose policies by which users and nodes can interact with each other in order to achieve good network behaviour.

6.3 User-Node Interaction Mechanism

We now propose simple but powerful policies by which users and nodes can interact with each other in order to achieve good network behaviour. Specifically, we propose allocation strategies for users with the objective of minimising the probability of experiencing bad QoS when connecting to a given node. Additionally, we propose a policy for nodes to adapt their fees and incentivise users to reduce their demand in the presence of congestion. Our policies generally operate as follows.

- Nodes measure the expected delay that is associated with processing a transaction. Nodes may additionally require a fee for their service that can be adapted in response to congestion, and this can also be provided as a QoS indicator for users.

- Users gather these QoS indicators provided by the nodes and construct a vector p representing a probability distribution over the nodes. Users then sample this distribution to select a node based on a probability that is proportional to the QoS indicator (inversely proportional to expected delay/fees) of a given node.

In Figure 6.2, we show an example of this user-node interaction in which users directly query nodes for their QoS indicators. In practice, nodes would most likely broadcast these values by writing to a commonly accessible resource for the users to query which could provide a more efficient solution.

Stochastic policies of this nature have been studied and analysed in the context of transportation networks [85, 86, 87]. We shall not repeat this analysis here. Intuitively, when all users operate this policy, nodes are kept busy, and users minimise their probability of experiencing a large delay. While such policies operate well in transportation and mobility networks, a defining characteristic of DLT based environments is that they are adversarial. Our mechanism can be adapted to deal with an adversarial setting by requiring users to complete a small PoW to prevent spamming or by introducing fees as we shall demonstrate in Section 6.3.3.

Another consideration worth discussing is whether or not we can expect users to follow such policies, and this can not be determined definitively until implemented in a real system with human actors involved. However, the choice faced by users can be intuitively thought of in a similar manner to the choice faced by network users in TCP—following the stochastic policy is an unstable Nash equilibrium in which an individual user could benefit from deviating from the protocol, but if all users defected then they would all be worse off. Just as in TCP, if the user's choice can be abstracted away effectively and the default option is set to our stochastic policy, it is reasonable to expect that most users will not deviate from this.

In the algorithms outlined in this section, $\mathcal{N}$ is the set of all available nodes, p is a vector whose i^{th} element, p_i, represents the probability of selecting node i, and rep_i is node i's reputation. In this work, we assume that $\mathcal{N}$ includes all nodes, i.e., all nodes make their service available to all users. In reality, some nodes may not wish to make their services available to users or may even make them available to a select group of users, perhaps on a subscription basis. However, we defer further considerations of such complexities to future work.

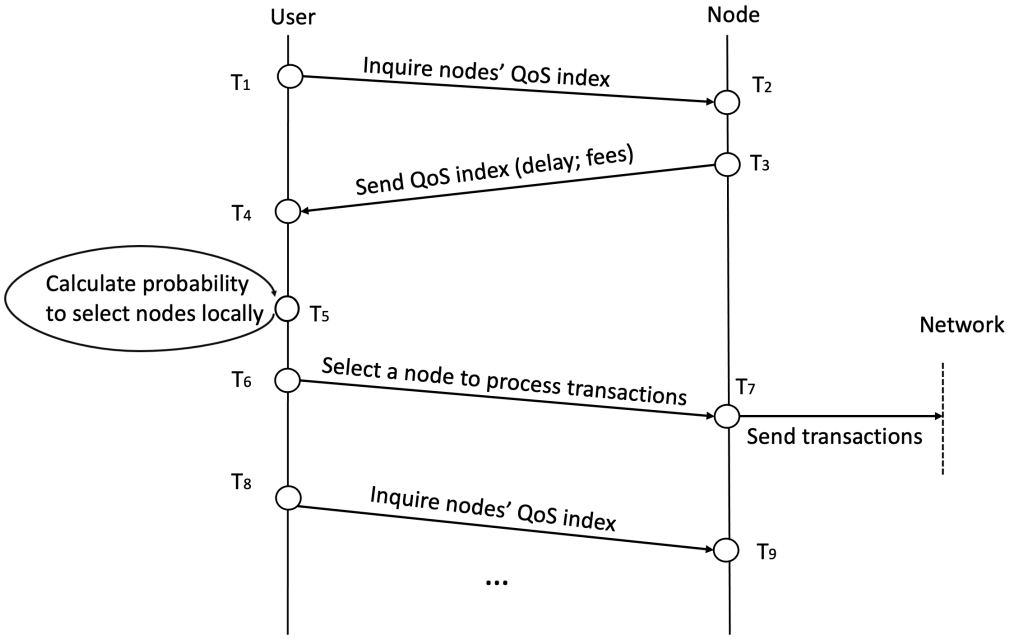


Figure 6.2: User-node interaction mechanism

6.3.1 Naive Policies

To provide a benchmark for comparison to the policies we will propose, we begin by describing two stochastic policies that may seem reasonable. We refer to these as *naive* policies as they do not attempt in any serious manner to shape traffic reaching the nodes based on prevailing network conditions. It is important to reiterate that no state of the art exists in the context of DAG-based DLTs, as user-node interaction usually refers to blockchain where a common mempool is used. The first policy we describe is a simple *uniform random node selection* (URNS) policy, as outlined in Algorithm 9. The second, less naive, policy which we will compare to our proposed policies is referred to as *reputation-based node selection* (RBNS), as described in Algorithm 10.

Algorithm 9 URNS executed by users

1: $p_j \leftarrow 1/|\mathcal{N}|, \forall j \in \mathcal{N}$ ▷ Initialise probabilities

Repeat each time a transaction is sent:
2: $j \leftarrow$ random sample from p
3: Send transactions to node j

Algorithm 10 RBNS executed by users

1: $p_j = \frac{rep_j}{\sum_{i \in \mathcal{N}} rep_i}, \forall j \in \mathcal{N}$ ▷ Initialise probablities

Repeat each time a transaction is sent:
2: $j \leftarrow$random sample from p
3: Send transactions to node j

As can be seen from these algorithms, URNS involves selecting nodes uniformly at random, without taking into account any information about the node whatsoever. RBNS, on the other hand, makes use of the globally known reputation of each node which gives some information about the rate at which each node can issue new transactions. Both URNS and RBNS are *open loop* policies in the sense that they do not make use of feedback from nodes about the current QoS indicators such as delay as we do in the policies which we propose next. As we shall see in Section 6.4, closed loop policies based on measured QoS metrics offer significant improvement over

these open loop alternatives.

6.3.2 Selection Policy Based on Delay

In order to improve upon the naive policies stated above, we would like to introduce QoS metrics from nodes. We first focus our attention on delay as the primary indicator of QoS. As such, we refer to our solution as *delay-based node selection* (DBNS) as presented in Algorithm 11.

Algorithm 11 DBNS executed by users

Repeat each time a transaction is sent:

1: $p_j \leftarrow \frac{rep_j/\tau_j}{\sum_{i\in\mathcal{N}} rep_i/\tau_i}, \forall j \in \mathcal{N}$ ▷ Update probabilities
2: $j \leftarrow$ random sample from p
3: Send transaction to node j

Algorithm 11 is executed by a user each time it wishes to have a transaction issued to the ledger. The probability update of line 1 is the critical step of the algorithm, namely:

$$p_j = \frac{rep_j/\tau_j}{\sum_{i\in\mathcal{N}} rep_i/\tau_i}$$

where p_j is the probability that a user chooses node j and τ_i is the expected delay in node j's LTP. Note that this policy is decentralised, requiring no information about the operation of other users. The only information required by users from each node is their reputation and their expected delay. Moreover, recall that the reputation of each node is a quantity that is known by all users.

The expected delay, on the other hand, can be calculated by nodes at regular intervals and broadcast to all users. The delay signal, τ_i, broadcast by nodes acts as a feedback to users about the state of the network, resulting in a closed loop system. The expected delay for a transaction arriving to node i's LTP at time t can be estimated as follows:

$$\tau_i(t) = \frac{L_i(t)}{\tilde{\lambda}_i(t)} \tag{6.1}$$

where $L_i(t)$ is the number of transactions in node i's LTP at time t (including the newly arrived transaction). $\tilde{\lambda}_i(t)$ is the expected average service rate of the LTP over the time this new transaction spends waiting to be issued. In a PoW ledger, this would be a fixed value, $\tilde{\lambda}_i(t) = \lambda_i$, based on the computing power of node i (provided the node's computing power remains constant). In this work, however, we focus our attention on the IOTA protocol in which the service rate of the LTP is governed by an AIMD rate setter as described in Chapter 4. We employ a moving average filter to estimate the issue rate of each node.

6.3.3 Including Transaction Fees

The above policy can be extended to include transaction fees which offers improved security for nodes against malicious users and demonstrates the ability of our approach to capture broader notions of QoS. This extension requires a policy for nodes as well as for users. Specifically, nodes require a fee σ_i to be included in any transaction and they broadcast this along with their expected delay τ_i, to users. When congestion is detected by nodes, they increase their fee to reduce the load on themselves. To achieve this goal, a simple P controller (it also can be adapted to PI/PID equivalent) is adopted here for node i.

$$\sigma_i(t) = \max\left(0, K_{pi}\left(\tau_i(t) - r_i\right)\right) \tag{6.2}$$

where K_{pi} is the proportional gain (a positive constant) and r_i is a delay setpoint. If the expected delay of node i's LTP, $\tau_i(t)$, is less that r_i, then node i will set their fee to zero, and as delay increases beyond this setpoint, the fee $\sigma_i(t)$ will increase at a rate proportional to K_{pi}. Both K_{pi} and r_i can be tuned by nodes on an individual basis, but we will assume that they are equal for all nodes in this work.

Note that this policy does not require nodes to be altruistic. On the contrary, if parameters are chosen appropriately, we intuitively expect this policy to maximise a node's income from fees.

This is because the policy maintains some LTP delay which implies that some transactions are maintained in the node's LTP, and furthermore, the fee for each of these transactions is set as high as possible without deterring too many users and ending up with an empty LTP. For this reason, we can expect that nodes would follow this policy, or some similar policy optimised for their needs, out of their own selfish desire to make profit.

Meanwhile, users can compute a cost function based on their desired trade-off between fee and delay to get a QoS metric for each node. The cost function for user m can be specified as follows

$$c_{mi}(t) = a_m \tau_i(t) + (1 - a_m)\, \sigma_i(t)$$

where $c_{mi}(t)$ is the QoS metric used by user m for node i at time t, a_m determines the weights assigned to delay $\tau_i(t)$ and fees $\sigma_i(t)$ for user m. Larger values for a_m indicate that user m cares greatly about having low delay whilst lower values reflect that this user is more concerned about paying low fees. Users also define a cost threshold, $c_m^{\max}$, above which this user will not continue sending transactions, which means when congestion increases, fees go up and some users stop sending transactions until the QoS metric is lower than their threshold. The algorithm employed by users, which we call DBNS+, including both delay and fees, is described in Algorithm 12.

Algorithm 12 DBNS+ executed by user m

Repeat each time a transaction is sent:
$\mathcal{N}^* \leftarrow \mathcal{N}$
for $j \in \mathcal{N}$ **do**
  $c_{mj} = a_m \tau_i + (1 - a_m)\sigma_i$
  if $c_{mj} > c_m^{\max}$ **then**
    remove j from $\mathcal{N}^*$
  end if
end for
if $\mathcal{N}^*$ is not empty **then**
  $p_j \leftarrow \frac{rep_j/c_{mj}}{\sum_{i \in \mathcal{N}^*} rep_i/c_{mi}}, \forall j \in \mathcal{N}^*$
  $j \leftarrow$ random sample from p
  Send transaction to node j
end if

6.4 Simulations

In the following experiments, 50 nodes are available to issue transactions for user, where each node in the network is peered with 4 randomly chosen neighbours. The nodes we discuss in this chapter are obeying restrictions imposed by the access control algorithm of Chapter 4 wherein the issue rates of nodes are controlled by an AIMD algorithm. Transactions issued by nodes need to be forwarded to neighbours to achieve a shared view of the ledger. As described in Chapter 4, the total rate at which nodes can process new transactions is given by ν. In the below simulations, ν is set to 50 transactions per second. All simulations are 3 minutes of simulation time, and results are averaged over 10 Monte Carlo simulations, unless otherwise specified. The reputation distribution of the nodes follows a Zipf distribution with exponent 0.9. These statistics are chosen to be representative of real values that have been measured from the IOTA network[26].

We organise our simulation results as follows. Experiments were conducted by changing the node selection policy employed by users from the portfolio of stochastic policies: (i) URNS which selects nodes uniformly at random (see Algorithm 9); (ii) RBNS which selects nodes based on their reputation (see Algorithm 10); (iii) DBNS which selects nodes based on their reputation and delay of their service (see Algorithm 11); (iv) DBNS+ which builds on DBNS by including transaction fees (see Algorithm 12). Apart from changing the node selection policy employed by users, all other simulation parameters are the same in each scenario.

For each group of simulations (i)–(iv), we consider three different scenarios based on the combined sending rate of users which represents the total traffic which nodes must process. In all scenarios,

[26]This model is well suited to reputation systems derived from wealth, i.e., PoS[45].

each new transaction is generated by a new user, and these transactions follow a Poisson arrival process. Each new user is independent of all others and they select a node according to the policy being tested. Each policy is tested in isolation, i.e., we do not consider mixtures of different policies from different users in this work.

- In scenario (a), users send transactions at an average rate that is 90% of the total network scheduling rate. Note that as transactions do not arrive uniformly over time (they follow a Poisson arrival process), the instantaneous arrival rate will sometimes exceed the scheduling rate of the network.

- In scenario (b), we consider an average sending rate at 98% of the total network scheduling rate to demonstrate behaviour very close to full capacity.

- Finally, in scenario (c), we set the average sending rate to 120% to simulate how these approaches deal with congestion in high-load situations above the network capacity.

Summary of Findings

To compare each policy, we consider the delays experienced by users in the LTP of their chosen node.

(i) URNS policy: for each scenario, (a)–(c), Figure 6.3 plots traces for the delays experienced at each of the 50 nodes, where the thickness of each trace is proportional to the reputation of the node. We observe that lower reputation nodes experience severe delays and high reputation nodes are underutilised and hence experience very low delays. Figure 6.4 shows a zoomed in view of the same data. From the perspective of nodes, traffic from users is very poorly balanced, resulting in low reputation nodes being overwhelmed and high reputation nodes being underutilised.

(ii) RBNS policy: in Figure 6.5, which shows the LTP delays of nodes in the RBNS simulations, the improvement over the URNS policy is immediately evident. The plots once again show traces for the delays experienced at each of the 50 nodes, where the thickness of each trace is proportional to the reputation of the node. We can see from these plots that at 98% capacity, fluctuations in delay of some nodes become more evident due to the fact that no feedback is used here. In the high load scenario of 120% capacity, we see that higher reputation nodes experience higher delays which is related to the fact that the issue rates of nodes are not perfectly fair with respect to their reputation. It is clear that a closed loop approach is required to achieve better fairness.

(iii) DBNS policy: Figure 6.6 again illustrates the delays experienced at each of the 50 nodes, where the thickness of each trace is proportional to the reputation of the node, this time for the DBNS policy. We see that there is less fluctuation in scenario (a) and (b) than was the case for RBNS, and in case (c) we observe that the traffic is balanced more fairly among nodes due to the use of feedback in DBNS. However, there is still no policy in place to control congestion in the LTP effectively so DBNS still performs poorly in scenario (c) with the delay continuing to grow throughout simulation.

(iv) DBNS+ policy: Finally, we present preliminary results for DBNS+. The proportional gain K_{pi} is set to 0.8, and desired level of delay $r_i(t)$) is set to 15 for all nodes. a_m is set to 0.6 for all users and $c_m^{\max}$ is set randomly between 8 and 10 for each user to capture some variation in what users expect in terms of delay and fees. As can be seen from the plots in Figure 6.7, in scenarios (a) and (b), DBNS+ performs similarly to DBNS, but when we introduce more severe congestion in scenario (c), there are clear advantages to the DBNS+ policy. In particular, we see in scenario (c) that the delay of each node is effectively stabilised by DBNS+. The reason for the stabilisation of delays after a certain point is that nodes increase their fees in response to congestion and the QoS metric, $c_{mi}(t)$, for some users eventually exceeds the threshold $c_m^{\max}$. At this point, users stop sending transactions to node i. Note that the oscillations evident in the plots for scenario (c) in Figure 6.7 are a direct result of the AIMD algorithm governing the service rate of the LTP which in turn determines the estimated delay serving as feedback to users.

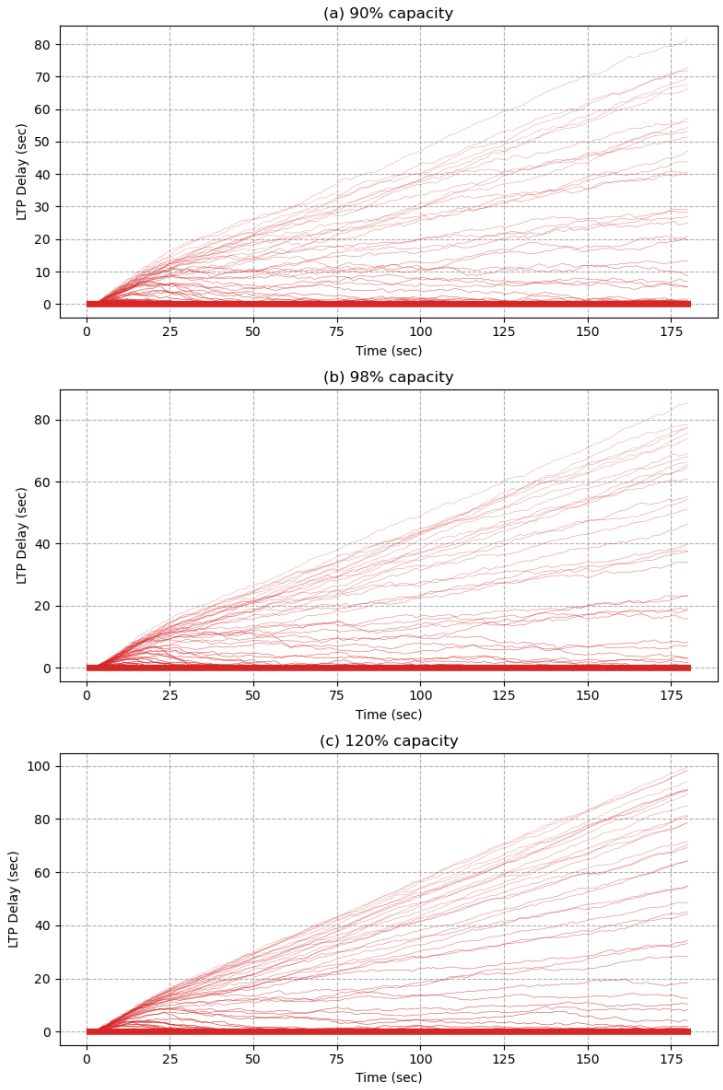


Figure 6.3: Uniform random node selection (URNS): delays experienced in the LTP of each node.

The results provided by Figures 6.3–6.7 primarily focus on the perspective of nodes rather than directly considering the experience of users. To capture the user perspective more effectively, we can consider the statistics provided by Table 6.1 which gives the expected delay experiences by a user and Table 6.2 which provides the probability that a user experiences a delay of greater than 20 seconds. This latter statistic is of interest because some users may not care about the exact delay of their transaction as long as does not exceed some threshold that they deem unreasonable. The threshold of 20 seconds applies well to the scenarios considered here but a different acceptable threshold may be considered depending on the needs of users in a given application. This statistic can be thought of as the probability that a random user will get unlucky and have a bad experience, which may turn them off using the technology again in the future.

Table 6.1 confirms that URNS performs poorly in all scenarios from the perspective of a user with expected delays higher across the board. RBNS, DBNS, and DBNS+ all perform well from the perspective of a user when demand is at 90% and 98%, as we see from the expected delays below 1 second in these scenarios. In the high load scenario (c), we only see a slight improvement in each more advanced policy, with the expected delay of DBNS+ just over

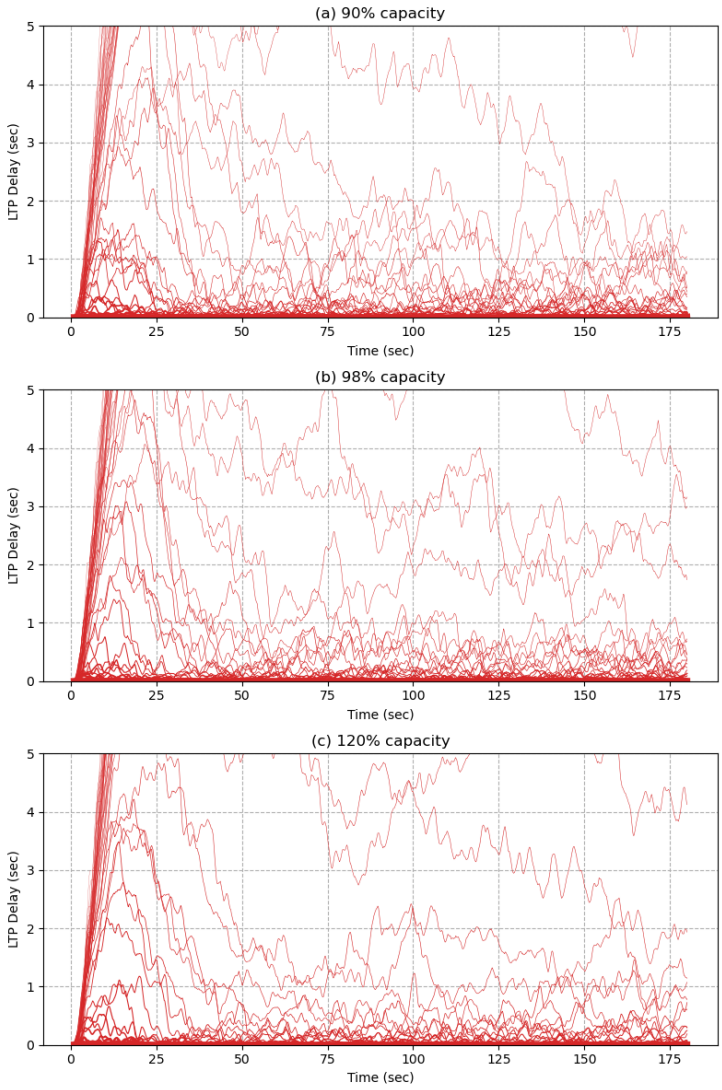


Figure 6.4: Uniform random node selection (URNS): zoomed in view of delays experienced in the LTP of each node.

a 1 second less than that of DBNS. However, we expect that if these simulation were run for longer, we would see a larger difference due to the growing LTP delays in DBNS as we observed in Figure 6.6.

Table 6.2 tells a similar story—the probability of being unlucky (experiencing a delay greater than 20 seconds) in URNS is high for users in all scenarios. RBNS, DBNS and DBNS+ all operate very well in this regard in scenarios (a) and (b), with no instances of delays greater than 20 seconds recorded during the simulations of these scenarios. The high load scenario (c) reveals the improvement offered by DBNS+ from the perspective of users, with a probability of 0.01 of experiencing a delay greater than 20 seconds. It is worth noting once again that the threshold of 20 seconds is selected here because it applies well to the scenarios and parameters chosen, but some threshold related to a meaningful delay should be chosen when evaluating these policies in practice.

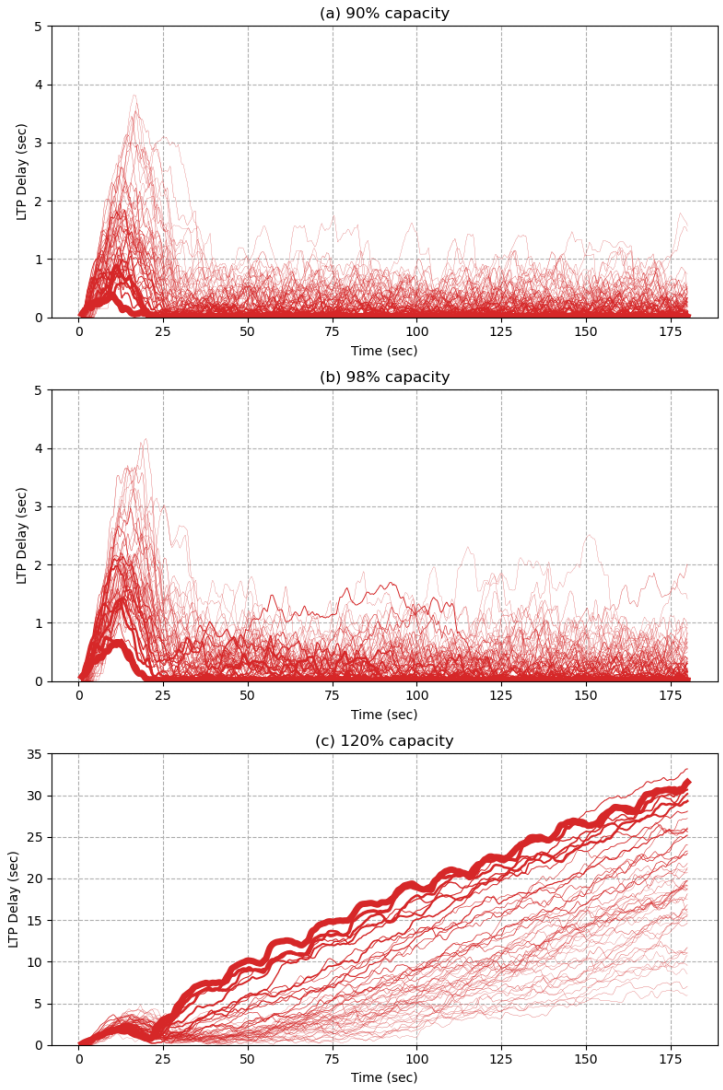


Figure 6.5: Reputation-based node selection (RBNS): delays experienced in the LTP of each node.

6.5 Chapter Summary

In this chapter, we proposed a user-node interaction mechanism for DAG-based DLTs which seeks to improve the usability of such networks for basic users who do not wish to run a fully operation node. The mechanism involves nodes calculating QoS metrics and providing this information to users. Users then use this feedback in a stochastic policy to select a node. Experiments were carried out to validate the effectiveness of the proposed algorithms. In particular, our experiments show that by combining fees and measurements of expected delays, QoS can be provided in a fair manner to users and the demand from users can be allocated fairly among nodes.

There are a number of simplifying assumptions used in this work, for example, we assumed that users have full knowledge of nodes statistics, which is probably unfeasible for large networks. For future work, it would be interesting to study how our policies perform when users query only a subset of nodes for QoS indicators. It would also be interesting to consider heterogenous user behaviour and to introduce more complex node behaviours such as collaboration between nodes and sharing of fees through mechanisms such as Shapley value [102]. Other work to be considered includes attack analyses for the policies presented here.

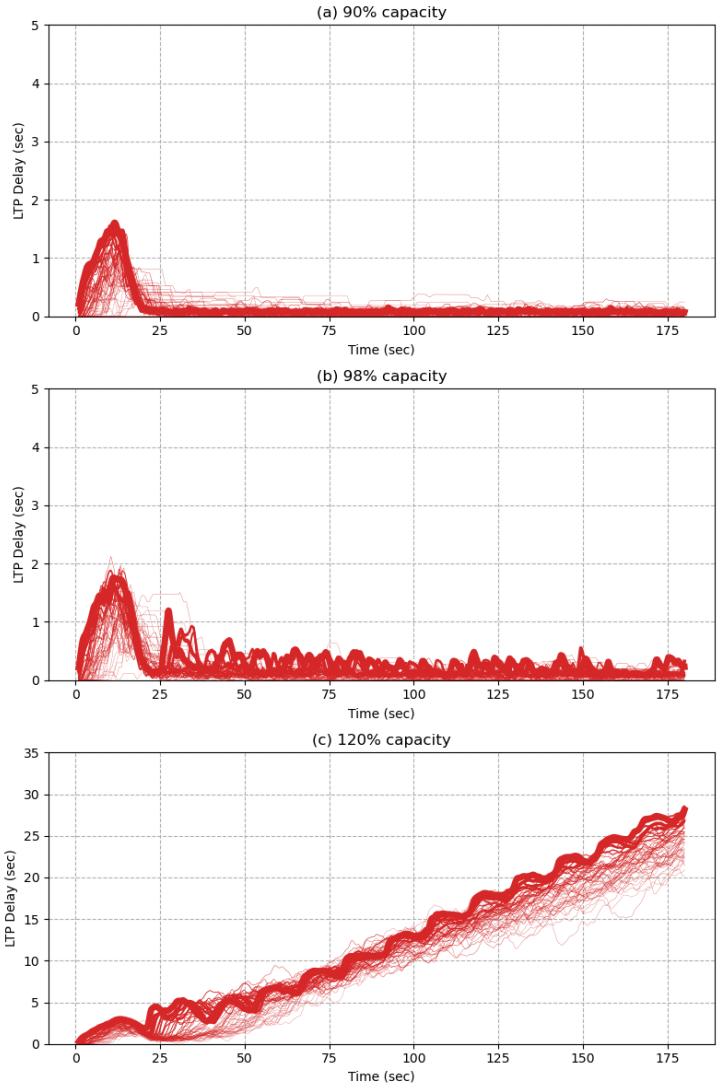


Figure 6.6: Delay-based node selection (DBNS): delays experienced in the LTP of each node.

Table 6.1: Expected LTP delay

	(a) 90% capacity	(b) 98% capacity	(c) 120% capacity
URNS	10.83 s	12.84 s	17.24 s
RBNS	0.29 s	0.40 s	11.80 s
DBNS	0.15 s	0.26 s	11.58 s
DBNS+	0.15 s	0.21 s	10.17 s

Table 6.2: Probability of LTP delay greater than 20 seconds.

	(a) 90% capacity	(b) 98% capacity	(c) 120% capacity
URNS	0.21	0.25	0.32
RBNS	0.00	0.00	0.25
DBNS	0.00	0.00	0.20
DBNS+	0.00	0.00	0.01

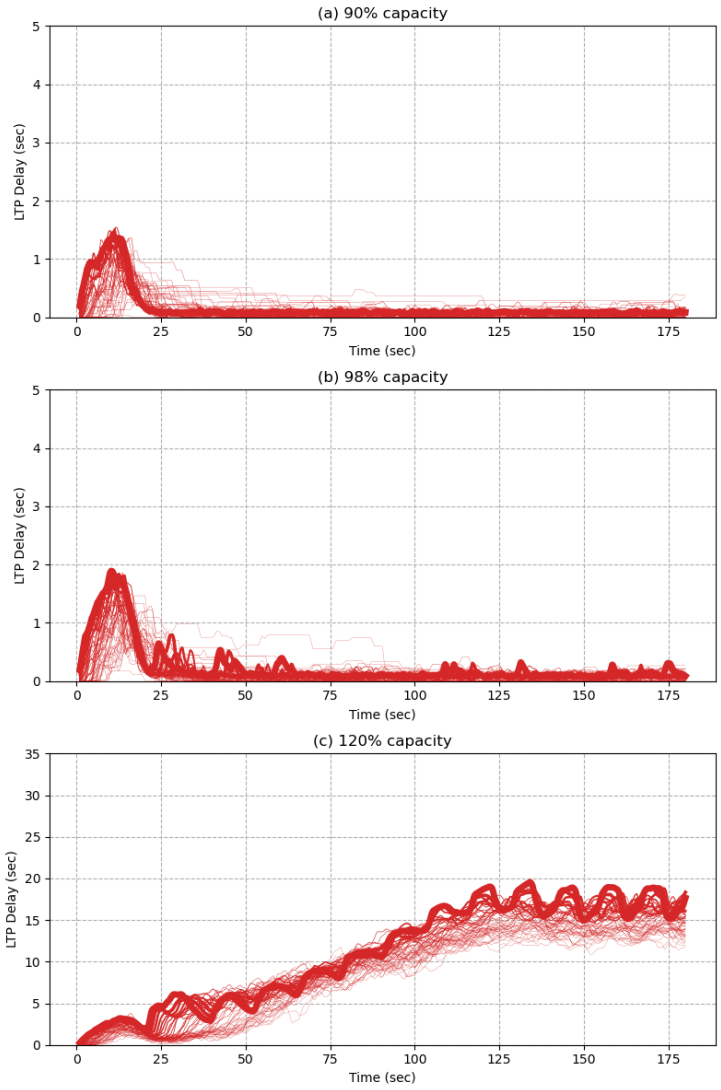


Figure 6.7: Delay-based node selection with fees (DBNS+): delays experienced in the LTP of each node.

Bibliography

[1] P. Ferraro, C. King, and R. Shorten, "Distributed ledger technology for smart cities, the sharing economy, and social compliance," *IEEE Access*, vol. 6, pp. 62 728–62 746, 2018.

[2] S. Nakamoto, "Bitcoin: A peer-to-peer electronic cash system," 2008. [Online]. Available: https://bitcoin.org/bitcoin.pdf

[3] M. Drijvers, S. Gorbunov, G. Neven, and H. Wee, "Pixel: Multi-signatures for consensus." in *USENIX Security Symposium*, 2020, pp. 2093–2110.

[4] Y. Gilad, R. Hemo, S. Micali, G. Vlachos, and N. Zeldovich, "Algorand: Scaling Byzantine agreements for cryptocurrencies," in *Proceedings of the 26th Symposium on Operating Systems Principles*, 2017, pp. 51–68.

[5] E. Deirmentzoglou, G. Papakyriakopoulos, and C. Patsakis, "A survey on long-range attacks for proof of stake protocols," *IEEE Access*, vol. 7, pp. 28 712–28 725, 2019.

[6] T. Hanke, M. Movahedi, and D. Williams, "DFINITY technology overview series, consensus system," 2018. [Online]. Available: arXivpreprintarXiv:1805.04548

[7] A. Cullen, P. Ferraro, R. Shorten, W. Sanders, and L. Vigneri, "Access control in adversarial environments for IoT-oriented distributed ledgers," in *2021 IFIP/IEEE International Symposium on Integrated Network Management (IM)*. IEEE, 2021, pp. 968–973.

[8] A. Cullen, P. Ferraro, W. Sanders, L. Vigneri, and R. Shorten, "Access control for distributed ledgers in the internet of things: A networking approach," *IEEE Internet of Things Journal*, vol. 9, no. 3, pp. 2277–2292, 2022.

[9] E. Androulaki, A. Barger, V. Bortnikov, C. Cachin, K. Christidis, A. De Caro, D. Enyeart, C. Ferris, G. Laventman, Y. Manevich *et al.*, "Hyperledger fabric: a distributed operating system for permissioned blockchains," in *Proceedings of the thirteenth EuroSys conference*, 2018, pp. 1–15.

[10] M. Vukolić, "The quest for scalable blockchain fabric: Proof-of-work vs. BFT replication," in *International Workshop on Open Problems in Network Security*. Springer, 2015, pp. 112–125.

[11] M. C. Ballandies, M. M. Dapp, and E. Pournaras, "Decrypting distributed ledger design—taxonomy, classification and blockchain community evaluation," *Cluster Computing*, pp. 1–22, 2021.

[12] Q. Zhou, H. Huang, Z. Zheng, and J. Bian, "Solutions to scalability of blockchain: A survey," *IEEE Access*, vol. 8, pp. 16 440–16 455, 2020.

[13] Q. Wang, J. Yu, S. Chen, and Y. Xiang, "Sok: Diving into DAG-based blockchain systems," 2020. [Online]. Available: https://arxiv.org/abs/2012.06128

[14] T. K. Mackey, T.-T. Kuo, B. Gummadi, K. A. Clauson, G. Church, D. Grishin, K. Obbad, R. Barkovich, and M. Palombini, "'Fit-for-purpose?'–challenges and opportunities for appli-

cations of blockchain technology in the future of healthcare," *BMC Medicine*, vol. 17, no. 1, pp. 1–17, 2019.

[15] C. Decker and R. Wattenhofer, "Information propagation in the Bitcoin network," in *IEEE P2P 2013 Proceedings*. IEEE, 2013, pp. 1–10.

[16] C. Lee, "Litecoin," GitHub Repository, 2011. [Online]. Available: https://github.com/litecoin-project/litecoin

[17] BitCoin Cash, "Bitcoin Cash," GitHub Repository, 2017. [Online]. Available: https://github.com/bitcoin-cash-node/bitcoin-cash-node

[18] S. Inu, "Dogecoin," GitHub Repository, 2013. [Online]. Available: https://github.com/dogecoin/dogecoin

[19] Binance, "Binance chain whitepaper," 2020. [Online]. Available: https://github.com/bnb-chain/whitepaper

[20] Centre, "USD coin whitepaper," 2018. [Online]. Available: https://f.hubspotusercontent30.net/hubfs/9304636/PDF/centre-whitepaper.pdf

[21] A. Back, M. Corallo, L. Dashjr, M. Friedenbach, G. Maxwell, A. Miller, A. Poelstra, J. Timón, and P. Wuille, "Enabling blockchain innovations with pegged sidechains," 2014. [Online]. Available: https://blockstream.com/sidechains.pdf

[22] 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.

[23] 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.

[24] V. Buterin, "Ethereum whitepaper," 2014. [Online]. Available: https://ethereum.org/en/whitepaper/

[25] Y. Sompolinsky, Y. Lewenberg, and A. Zohar, "SPECTRE: A fast and scalable cryptocurrency protocol," 2016. [Online]. Available: https://eprint.iacr.org/2016/1159.pdf

[26] S. Popov, "The tangle (IOTA whitepaper)," 2016. [Online]. Available: http://www.descryptions.com/Iota.pdf

[27] L. Baird, M. Harmon, and P. Madsen, "Hedera: A public hashgraph network and governing council," 2020. [Online]. Available: https://hedera.com/hh_whitepaper_v2.1-20200815.pdf

[28] I. Bentov, P. Hubáček, T. Moran, and A. Nadler, "Tortoise and hares consensus: the meshcash framework for incentive-compatible, scalable cryptocurrencies," in *International Symposium on Cyber Security Cryptography and Machine Learning*. Springer, 2021, pp. 114–127.

[29] A. Churyumov, "Byteball: A decentralized system for storage and transfer of value," 2016. [Online]. Available: https://byteball.org/Byteball.pdf

[30] C. LeMahieu, "Nano whitepaper," 2022. [Online]. Available: https://docs.nano.org/living-whitepaper/

[31] P. Ferraro, C. King, and R. Shorten, "On the stability of unverified transactions in a DAG-based distributed ledger," *IEEE Transactions on Automatic Control*, vol. 65, no. 9, pp. 3772–3783, 2019.

[32] B. Kuśmierz, W. Sanders, A. Penzkofer, A. Capossele, and A. Gal, "Properties of the tangle for uniform random and random walk tip selection," in *2019 IEEE International Conference*

on Blockchain (Blockchain). IEEE, 2019, pp. 228–236.

[33] G. Bu, Ö. Gürcan, and M. Potop-Butucaru, "G-IOTA: Fair and confidence aware tangle," in *IEEE INFOCOM 2019-IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. IEEE, 2019, pp. 644–649.

[34] S. Popov, O. Saa, and P. Finardi, "Equilibria in the tangle," *Computers & Industrial Engineering*, vol. 136, pp. 160–172, 2019.

[35] Y. Li, B. Cao, M. Peng, L. Zhang, L. Zhang, D. Feng, and J. Yu, "Direct acyclic graph-based ledger for internet of things: Performance and security analysis," *IEEE/ACM Transactions on Networking*, vol. 28, no. 4, pp. 1643–1656, 2020.

[36] A. Penzkofer, O. Saa, and D. Dziubałtowska, "Impact of delay classes on the data structure in IOTA," in *Data Privacy Management, Cryptocurrencies and Blockchain Technology*. Springer, 2021, pp. 289–300.

[37] A. Penzkofer, B. Kuśmierz, A. Capossele, W. Sanders, and O. Saa, "Parasite chain detection in the IOTA protocol," in *2nd International Conference on Blockchain Economics, Security and Protocols*, 2021.

[38] K. J. O'Dwyer and D. Malone, "Bitcoin mining and its energy footprint," in *Proceedings of 25th IET Irish Signals and Systems Conference*, 2014.

[39] 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.

[40] V. Buterin and V. Griffith, "Casper the friendly finality gadget," *arXiv preprint arXiv:1710.09437*, 2017.

[41] J. Kwon, "Tendermint: Consensus without mining," 2014. [Online]. Available: https://tendermint.com/static/docs/tendermint.pdf

[42] DFINITY Team, "The internet computer for geeks," 2022. [Online]. Available: https://dfinity.org/whitepaper.pdf

[43] E. Deirmentzoglou, G. Papakyriakopoulos, and C. Patsakis, "A survey on long-range attacks for proof of stake protocols," *IEEE Access*, vol. 7, pp. 28 712–28 725, 2019.

[44] S. Popov, H. Moog, D. Camargo, A. Capossele, V. Dimitrov, A. Gal, A. Greve, B. Kuśmierz, S. Müller, A. Penzkofer *et al.*, "The coordicide," 2020. [Online]. Available: https://files.iota.org/papers/Coordicide_WP.pdf

[45] C. I. Jones, "Pareto and Piketty: The macroeconomics of top income and wealth inequality," *Journal of Economic Perspectives*, vol. 29, no. 1, pp. 29–46, 2015.

[46] S. Popov and W. J. Buchanan, "FPC-BI: Fast probabilistic consensus within Byzantine infrastructures," *Journal of Parallel and Distributed Computing*, vol. 147, pp. 77–86, 2021.

[47] S. Popov and S. Müller, "Voting-based probabilistic consensuses and their applications in distributed ledgers," *Annals of Telecommunications*, vol. 77, no. 1, pp. 77–99, 2022.

[48] B. Kuśmierz, S. Müller, and A. Capossele, "Committee selection in DAG distributed ledgers and applications," in *Intelligent Computing*. Springer, 2021, pp. 840–857.

[49] S. E. Chang and Y. Chen, "When blockchain meets supply chain: A systematic literature review on current development and potential applications," *IEEE Access*, vol. 8, pp. 62 478–62 494, 2020.

[50] J. Brogan, I. Baskaran, and N. Ramachandran, "Authenticating health activity data using

distributed ledger technologies," *Computational and structural biotechnology journal*, vol. 16, pp. 257–266, 2018.

[51] H.-N. Dai, M. Imran, and N. Haider, "Blockchain-enabled internet of medical things to combat covid-19," *IEEE Internet of Things Magazine*, vol. 3, no. 3, pp. 52–57, 2020.

[52] R. Overko, R. Ordóñez-Hurtado, S. Zhuk, P. Ferraro, A. Cullen, and R. Shorten, "Spatial positioning token (SPToken) for smart mobility," *IEEE Transactions on Intelligent Transportation Systems*, vol. 23, no. 2, pp. 1529–1542, 2022.

[53] A. Cullen, P. Ferraro, C. King, and R. Shorten, "Distributed ledger technology for smart mobility: Variable delay models," in *2019 IEEE 58th Conference on Decision and Control (CDC)*, 2019, pp. 8447–8452.

[54] D. Easley, M. O'Hara, and S. Basu, "From mining to markets: The evolution of Bitcoin transaction fees," *Journal of Financial Economics*, vol. 134, no. 1, pp. 91–109, 2019.

[55] A. Cullen, P. Ferraro, C. King, and R. Shorten, "On the resilience of DAG-based distributed ledgers in IoT applications," *IEEE Internet of Things Journal*, 2020.

[56] IOTA Foundation, "GoShimmer," GitHub Repository, 2020. [Online]. Available: https://github.com/iotaledger/goshimmer

[57] J. R. Douceur, "The Sybil attack," in *International workshop on peer-to-peer systems*. Springer, 2002, pp. 251–260.

[58] J. A. Kroll, I. C. Davey, and E. W. Felten, "The economics of Bitcoin mining, or Bitcoin in the presence of adversaries," in *Proceedings of WEIS*, 2013, p. 11.

[59] J. Huang, L. Kong, G. Chen, M.-Y. Wu, X. Liu, and P. Zeng, "Towards secure industrial IoT: Blockchain system with credit-based consensus mechanism," *IEEE Transactions on Industrial Informatics*, vol. 15, no. 6, pp. 3680–3689, 2019.

[60] L. Vigneri and W. Welz, "On the fairness of distributed ledger technologies for the internet of things," in *2020 IEEE International Conference on Blockchain and Cryptocurrency (ICBC)*. IEEE, 2020, pp. 1–3.

[61] S. Biswas, K. Sharif, F. Li, S. Maharjan, S. P. Mohanty, and Y. Wang, "PoBT: A lightweight consensus algorithm for scalable IoT business blockchain," *IEEE Internet of Things Journal*, vol. 7, no. 3, pp. 2343–2355, 2020.

[62] Y. Lewenberg, Y. Sompolinsky, and A. Zohar, "Inclusive block chain protocols," in *International Conference on Financial Cryptography and Data Security*. Springer, 2015, pp. 528–547.

[63] 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.

[64] R. Van Renesse, D. Dumitriu, V. Gough, and C. Thomas, "Efficient reconciliation and flow control for anti-entropy protocols," in *proceedings of the 2nd Workshop on Large-Scale Distributed Systems and Middleware*, 2008, pp. 1–7.

[65] D. Grossman *et al.*, "New terminology and clarifications for DiffServ," in *RFC 3260, April*, 2002.

[66] Y. Bernet, P. Ford, R. Yavatkar, F. Baker, L. Zhang, M. Speer, R. Braden, B. Davie, J. Wroclawski, and E. Felstaine, "A framework for integrated services operation over DiffServ networks," in *RFC 2998, November*, 2000.

[67] J. Nagle, "On packet switches with infinite storage," *IEEE Transactions on Communications*,

vol. 35, no. 4, pp. 435–438, 1987.

[68] A. Demers, S. Keshav, and S. Shenker, "Analysis and simulation of a fair queueing algorithm," *Internetworking: Research and experience*, vol. 1, no. 1, pp. 3–26, 1990.

[69] M. Shreedhar and G. Varghese, "Efficient fair queuing using deficit round-robin," *IEEE/ACM Transactions on Networking*, vol. 4, no. 3, pp. 375–385, 1996.

[70] M. H. MacGregor and W. Shi, "Deficits for bursty latency-critical flows: DRR++," in *Proceedings IEEE International Conference on Networks 2000 (ICON 2000). Networking Trends and Challenges in the New Millennium.* IEEE, 2000, pp. 287–293.

[71] J. Postel *et al.*, "Transmission control protocol," in *RFC 793, September*, 1981.

[72] S. Floyd and V. Jacobson, "Random early detection gateways for congestion avoidance," *IEEE/ACM Transactions on Networking*, vol. 1, no. 4, pp. 397–413, 1993.

[73] V. Attias, L. Vigneri, and V. Dimitrov, "Preventing denial of service attacks in IoT networks through verifiable delay functions," in *IEEE GLOBECOM 2020*, 2020.

[74] S. Müller, A. Penzkofer, B. Kuśmierz, D. Camargo, and W. J. Buchanan, "Fast probabilistic consensus with weighted votes," in *Proceedings of the Future Technologies Conference (FTC) 2020, Volume 2.* Springer Nature, 2020, p. 360.

[75] T. Hoeiland-Joergensen, P. McKenney, D. Taht, J. Gettys, and E. Dumazet, "The flow queue CoDel packet scheduler and active queue management algorithm," in *RFC 8290*, 2018.

[76] M. Menth, M. Mehl, and S. Veith, "Deficit round robin with limited deficit savings (DRR-LDS) for fairness among TCP users," in *International Conference on Measurement, Modelling and Evaluation of Computing Systems.* Springer, 2018, pp. 188–201.

[77] J. Gettys and K. Nichols, "Bufferbloat: Dark buffers in the internet," *Queue*, vol. 9, no. 11, pp. 40–54, 2011.

[78] M. Corless, C. King, R. Shorten, and F. Wirth, *AIMD dynamics and distributed resource allocation.* SIAM, 2016.

[79] B. Suter, T. Lakshman, D. Stiliadis, and A. K. Choudhury, "Design considerations for supporting TCP with per-flow queueing," in *Proceedings of IEEE INFOCOM'98*, vol. 1, 1998, pp. 299–306.

[80] L. Zhao, L. Vigneri, A. Cullen, W. Sanders, P. Ferraro, and R. Shorten, "Secure access control for DAG-based distributed ledgers," *IEEE Internet of Things Journal, Early Access*, vol. 9, no. 13, pp. 10 792–10 806, 2022.

[81] M. Zichichi, S. Ferretti, and G. D'Angelo, "A distributed ledger based infrastructure for smart transportation system and social good," in *2020 IEEE 17th Annual Consumer Communications & Networking Conference (CCNC).* IEEE, 2020, pp. 1–6.

[82] M. N. Kamel Boulos, J. T. Wilson, and K. A. Clauson, "Geospatial blockchain: promises, challenges, and scenarios in health and healthcare," *International journal of health geographics*, vol. 17, no. 1, pp. 1–10, 2018.

[83] J. J. Hunhevicz and D. M. Hall, "Do you need a blockchain in construction? use case categories and decision framework for DLT design options," *Advanced Engineering Informatics*, vol. 45, p. 101094, 2020.

[84] E. Crisostomi, B. Ghaddar, F. Häusler, J. Naoum-Sawaya, G. Russo, and R. Shorten, *Analytics for the sharing economy: Mathematics, Engineering and Business perspectives.* Springer, 2020.

[85] E. D. Miller-Hooks and H. S. Mahmassani, "Least expected time paths in stochastic, time-varying transportation networks," *Transportation Science*, vol. 34, no. 2, pp. 198–215, 2000.

[86] H. Gehlot, H. Honnappa, and S. V. Ukkusuri, "An optimal control approach to day-to-day congestion pricing for stochastic transportation networks," *Computers & Operations Research*, vol. 119, p. 104929, 2020.

[87] C. F. Daganzo and Y. Sheffi, "On stochastic models of traffic assignment," *Transportation Science*, vol. 11, no. 3, pp. 253–274, 1977.

[88] M. Caliskan, D. Graupner, and M. Mauve, "Decentralized discovery of free parking places," in *Proceedings of the 3rd international workshop on Vehicular ad hoc networks*, 2006, pp. 30–39.

[89] M. I. Idris, Y. Leng, E. Tamil, N. Noor, Z. Razak *et al.*, "Car park system: A review of smart parking system and its technology," *Information Technology Journal*, vol. 8, no. 2, pp. 101–113, 2009.

[90] A. Schlote, C. King, E. Crisostomi, and R. Shorten, "Delay-tolerant stochastic algorithms for parking space assignment," *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, no. 5, pp. 1922–1935, 2014.

[91] K. Baqer, D. Y. Huang, D. McCoy, and N. Weaver, "Stressing out: Bitcoin "stress testing"," in *International Conference on Financial Cryptography and Data Security*. Springer, 2016, pp. 3–18.

[92] F. Semchedine, L. Bouallouche-Medjkoune, and D. Aissani, "Task assignment policies in distributed server systems: A survey," *Journal of Network and Computer Applications*, vol. 34, no. 4, pp. 1123–1130, 2011.

[93] G. Ciardo, A. Riska, and E. Smirni, "Equiload: a load balancing policy for clustered web servers," *Performance Evaluation*, vol. 46, no. 2-3, pp. 101–124, 2001.

[94] M. Harchol-Balter, M. E. Crovella, and C. D. Murta, "On choosing a task assignment policy for a distributed server system," *Journal of Parallel and Distributed Computing*, vol. 59, no. 2, pp. 204–228, 1999.

[95] B. Schroeder and M. Harchol-Balter, "Evaluation of task assignment policies for supercomputing servers: The case for load unbalancing and fairness," *Cluster Computing*, vol. 7, no. 2, pp. 151–161, 2004.

[96] M. Colajanni, P. S. Yu, and D. M. Dias, "Analysis of task assignment policies in scalable distributed web-server systems," *IEEE Transactions on Parallel and Distributed Systems*, vol. 9, no. 6, pp. 585–600, 1998.

[97] D. L. Eager, E. D. Lazowska, and J. Zahorjan, "Adaptive load sharing in homogeneous distributed systems," *IEEE Transactions on Software Engineering*, no. 5, pp. 662–675, 1986.

[98] K. Ramamritham, J. A. Stankovic, and W. Zhao, "Distributed scheduling of tasks with deadlines and resource requirements," *IEEE Transactions on Computers*, vol. 38, no. 8, pp. 1110–1123, 1989.

[99] L. Cai, X. Shen, J. Pan, and J. W. Mark, "Performance analysis of TCP-friendly AIMD algorithms for multimedia applications," *IEEE Transactions on Multimedia*, vol. 7, no. 2, pp. 339–355, 2005.

[100] R. Shorten, F. Wirth, and D. Leith, "A positive systems model of TCP-like congestion control: asymptotic results," *IEEE/ACM Transactions on Networking*, vol. 14, no. 3, pp. 616–629, 2006.

[101] R. N. Shorten, D. J. Leith, J. Foy, and R. Kilduff, "Analysis and design of AIMD congestion

control algorithms in communication networks," *Automatica*, vol. 41, no. 4, pp. 725–730, 2005.

[102] A. M. Kharman, C. Jursitzky, Q. Zhao, P. Ferraro, J. Marecek, P. Pinson, and R. Shorten, "On the design of decentralised data markets," 2022. [Online]. Available: https://arxiv.org/abs/2206.06299

[103] A. Cullen, P. Ferraro, G. Russo, and R. Shorten, "Ad-hocChain: Cooperative sharing and trading infrastructure for electric vehicle charging networks," in *2019 IEEE Intelligent Transportation Systems Conference (ITSC)*, 2019, pp. 207–212.

[104] P. Richardson, D. Flynn, and A. Keane, "Impact assessment of varying penetrations of electric vehicles on low voltage distribution systems," in *IEEE PES general meeting*. IEEE, 2010, pp. 1–6.

[105] ——, "Optimal charging of electric vehicles in low-voltage distribution systems," *IEEE Transactions on Power Systems*, vol. 27, no. 1, pp. 268–279, 2011.

[106] S. Deilami, A. S. Masoum, P. S. Moses, and M. A. Masoum, "Real-time coordination of plug-in electric vehicle charging in smart grids to minimize power losses and improve voltage profile," *IEEE Transactions on Smart Grid*, vol. 2, no. 3, pp. 456–467, 2011.

[107] R. Shorten, J. O'Connell, B. Cardiff, G. Russo, P. Ferraro, and P. Cuffe, "Apparatus for directing power flow between multiple devices," U.S. Patent US 2020/0 376 969, 2017.

[108] E. Thompson, R. Ordóñez-Hurtado, W. Griggs, J. Y. Yu, B. Mulkeen, and R. Shorten, "On charge point anxiety and the sharing economy," in *2017 IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2017, pp. 1–6.

[109] J. O'Connell, B. Cardiff, and R. Shorten, "dockchain: A solution for electric vehicles charge point anxiety," in *2018 Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2018, pp. 1136–1142.

[110] International Electrotechnical Commission, "Electric vehicle conductive charging system—part 1: General requirements," 2017. [Online]. Available: https://webstore.iec.ch/publication/33644

[111] G. C. Buttazzo, *Hard real-time computing systems: predictable scheduling algorithms and applications*. Springer Science & Business Media, 2011, vol. 24.

[112] Irish Central Statistics Office, "Census of population 2016—profile 6 commuting in Ireland," 2016. [Online]. Available: https://www.cso.ie/en/releasesandpublications/ep/p-cp6ci/p6cii/p6td/

[113] D. Lazer, R. Kennedy, G. King, and A. Vespignani, "The parable of Google Flu: Traps in big data analysis," *Science*, vol. 343, pp. 1203–5, 2014.

[114] A. Sinha, D. Gleich, and K. Ramani, "Deconvolving feedback loops in recommender systems," in *Proceedings of the 30th International Conference on Neural Information Processing Systems (NIPS'16)*, Barcelona, Spain, 2016, pp. 3251–3259.

[115] E. Crisostomi, R. Shorten, and F. Wirth, "Smart cities: A golden age for control theory?" *IEEE Technology and Society Magazine*, vol. 35, no. 3, pp. 23–24, 2016.

[116] L. Bottou, J. Peters, J. Quiñonero Candela, D. X. Charles, D. M. Chickering, E. Portugaly, D. Ray, P. Simard, and E. Snelson, "Counterfactual reasoning and learning systems: The example of computational advertising," *Journal of Machine Learning Research*, vol. 14, no. 1, pp. 3207–3260, Jan. 2013.

[117] J. P. Epperlein, S. Zhuk, and R. Shorten, "Recovering markov models from closed-loop data," *Automatica*, vol. 103, pp. 116 – 125, 2019.

[118] R. Overko, R. Ordóñez-Hurtado, S. Zhuk, and R. Shorten, "Reinforcement learning augmented optimization for smart mobility," in *2019 IEEE 58th Conference on Decision and Control (CDC)*, 2019, pp. 1286–1292.

[119] R. S. Sutton and A. G. Barto, *Reinforcement learning: An introduction*. MIT press, 2018.

[120] L. Li, J. Liu, L. Cheng, S. Qiu, W. Wang, X. Zhang, and Z. Zhang, "CreditCoin: A privacy-preserving blockchain-based incentive announcement network for communications of smart vehicles." *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 7, pp. 2204–2220, 2018.

[121] Y. Yuan and F.-Y. Wang, "Towards blockchain-based intelligent transportation systems," in *IEEE Intelligent Transportation Systems Conference (ITSC)*, 11 2016, pp. 2663–2668.

[122] A. Dorri, M. Steger, S. S. Kanhere, and R. Jurdak, "Blockchain: A distributed solution to automotive security and privacy." *CoRR*, vol. abs/1704.00073, 2017.

[123] J. Krumm, "A Markov model for driver turn prediction," SAE Technical Paper, Tech. Rep., 2008.

[124] R. Simmons, B. Browning, Y. Zhang, and V. Sadekar, "Learning to predict driver route and destination intent," in *2006 IEEE Intelligent Transportation Systems Conference (ITSC)*, Sep. 2006, pp. 127–132.

[125] F. Belletti, D. Haziza, G. Gomes, and A. M. Bayen, "Expert level control of ramp metering based on multi-task deep reinforcement learning," *IEEE Transactions on Intelligent Transportation Systems*, vol. 19, no. 4, p. 1198–1207, 2017.

[126] L. Fridman, J. Terwilliger, and B. Jenik, "Deeptraffic: Crowdsourced hyperparameter tuning of deep reinforcement learning systems for multi-agent dense traffic navigation," 2019. [Online]. Available: https://arxiv.org/abs/1801.02805

[127] P. Mannion, J. Duggan, and E. Howley, "An experimental review of reinforcement learning algorithms for adaptive traffic signal control," *Autonomic Road Transport Support Systems. Springer*, pp. 47–66, 2016.

[128] M. O'Kelly, A. Sinha, H. Namkoong, J. Duchi, and R. Tedrake, "Scalable end-to-end autonomous vehicle testing via rare-event simulation," 2019. [Online]. Available: https://arxiv.org/abs/1811.00145

[129] S. El-Tantawy, B. Abdulhai, and H. Abdelgawad, "Multiagent reinforcement learning for integrated network of adaptive traffic signal controllers (MARLIN-ATSC): Methodology and large-scale application on downtown Toronto," *IEEE Transactions on Intelligent Transportation Systems*, vol. 14, no. 3, pp. 1140–1150, 2013.

[130] J. W. Vaughan, "Making better use of the crowd: How crowdsourcing can advance machine learning research," *Journal of Machine Learning Research*, vol. 18, no. 1, pp. 7026–7071, 2017.

[131] F. Hausler, E. Crisostomi, A. Schlote, I. Radusch, and R. Shorten, "Stochastic park-and-charge balancing for fully electric and plug-in hybrid vehicles," *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, pp. 895–901, 2014.

[132] A. Schlote, B. Chen, and R. Shorten, "On closed-loop bicycle availability prediction," *IEEE Transactions on Intelligent Transportation Systems*, vol. 15, pp. 1499–1455, 2015.

[133] H. R. Varian, "Causal inference in economics and marketing," *Proceedings of the National Academy of Sciences of the United States of America*, p. 7310–7315, 2016.

[134] D. Cosley, S. K. Lam, I. Albert, J. A. Konstan, and J. Riedl, "Is seeing believing?: How recommender system interfaces affect users' opinions," in *Proceedings of the SIGCHI Conference*

on Human Factors in Computing Systems. ACM, 2003, pp. 585–592.

[135] I. D. Loram, H. Gollee, M. Lakie, and P. J. Gawthrop, "Human control of an inverted pendulum: Is continuous control necessary? Is intermittent control effective? Is intermittent control physiological?" *The Journal of Physiology*, vol. 589, no. 2, pp. 307–324, 2011.

[136] C. Dann, T. Lattimore, and E. Brunskill, "Unifying PAC and regret: Uniform PAC bounds for episodic reinforcement learning," in *Advances in Neural Information Processing Systems*, 2017, pp. 5713–5723.

[137] J. F. T. Hastie, R. Tibshirani, *The Elements of Statistical Learning: Data Mining, Inference and Prediction*. Springer, 2001.

[138] P. A. Lopez, M. Behrisch, L. Bieker-Walz, J. Erdmann, Y.-P. Flötteröd, R. Hilbrich, L. Lücken, J. Rummel, P. Wagner, and E. WieBner, "Microscopic traffic simulation using sumo," in *IEEE Intelligent Transportation Systems Conference (ITSC)*. IEEE, 2018, pp. 2575–2582.

[139] E. W. Dijkstra, "A note on two problems in connexion with graphs," *Numerische Mathematik*, pp. 269–271, 1959.

[140] E. Syta, P. Jovanovic, E. K. Kogias, N. Gailly, L. Gasser, I. Khoffi, M. J. Fischer, and B. Ford, "Scalable bias-resistant distributed randomness," in *2017 IEEE Symposium on Security and Privacy (SP)*, 2017, pp. 444–460.

[141] R. Gennaro, S. Jarecki, H. Krawczyk, and T. Rabin, "Secure distributed key generation for discrete-log based cryptosystems," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 1999, pp. 295–310.

[142] DFINITY Team, "Threshold relay: How to achieve near-instant finality in public blockchains using a VRF," 2017. [Online]. Available: https://cryptorating.eu/whitepapers/DFINITY/threshold-relay-blockchain-stanford.pdf

[143] P. Schindler, A. Judmayer, N. Stifter, and E. Weippl, "Hydrand: Efficient continuous distributed randomness," 2018. [Online]. Available: https://eprint.iacr.org/2018/319.pdf

[144] A. K. Lenstra and B. Wesolowski, "A random zoo: sloth, unicorn, and trx." 2015. [Online]. Available: https://eprint.iacr.org/2015/366.pdf

[145] B. Bünz, S. Goldfeder, and J. Bonneau, "Proofs-of-delay and randomness beacons in ethereum," in *IEEE Security and Privacy on the blockchain (IEEE S&B)*, 2017. [Online]. Available: https://jbonneau.com/doc/BGB17-IEEESB-proof_of_delay_ethereum.pdf

[146] D. Boneh, B. Lynn, and H. Shacham, "Short signatures from the Weil pairing," in *International Conference on the Theory and Application of Cryptology and Information Security*. Springer, 2001, pp. 514–532.

[147] D. Boneh, C. Gentry, B. Lynn, and H. Shacham, "Aggregate and verifiably encrypted signatures from bilinear maps," in *International Conference on the Theory and Applications of Cryptographic Techniques*. Springer, 2003, pp. 416–432.

[148] C.-P. Schnorr, "Efficient signature generation by smart cards," *Journal of Cryptology*, vol. 4, no. 3, pp. 161–174, 1991.

Appendix A

DockChain: A Sharing Platform for Electric Vehicle Chargepoints

***Abstract*— Insufficient availability of charging facilities presents a major challenge for electric vehicle adoption as the automotive industry races to transition away from fossil fuels. The DLT-enabled sharing platform presented in this appendix seeks to alleviate this problem by improving the accessibility of existing charging infrastructure using a novel combination of specialised sharing hardware and a DLT-based platform for payment and vehicle-to-vehicle trading. Our solution, which we refer to as DockChain, allows multiple electric vehicles to connect to a single charge point and share its capacity in a fair and collaborative manner. DockChain is the subject of ongoing research and development alongside Go Eve, a company founded to commercialise the technology. Part of this work, which was completed in collaboration with Dr. Giovanni Russo of University College Dublin and Dr. Pietro Ferraro and Prof. Robert Shorten of Imperial College London, also appears in [103].**

Distributed ledgers are useful tools for untrusting agents to transact pseudo-anonymously with one another without the need for a trusted intermediary. As such, DLT is well suited for sharing applications such as the one presented in this appendix—DockChain. DockChain technology, depicted in Figure A.1, allows multiple electric vehicles (EVs) to connect to a single charge point and share the available power. Poor access to charging facilities is a major hindrance to large scale adoption of EVs, but scaling of charging infrastructure is far from straightforward due to the detrimental effects of high EV penetration on the grid [104]. It has been shown however that cooperative charging can mitigate the negative effects of EV charging loads [105], and indeed, can be beneficial in stabilising the grid in the presence of intermittent generation [106] although smart grid infrastructure to facilitate this is costly and time-consuming to implement. DockChain technology solves these problems by allowing available capacity of charge points to be shared in a cost-effective and straightforward manner.

The initial development of DockChain technology predates the start of this research [107, 108, 109], but a number of key innovations were developed as part of this PhD [103] which have unlocked new use cases and functionalities which would not otherwise have been possible. The most notable contributions of this PhD to the development of DockChain have been the following innovations:

- the first prototype of DockChain compatible with mode 3[27] public charge points, enabling applications where temporary and semi-permanent extension of existing charging infrastructure is required;
- a scheduling system and trading layer based on DLT for facilitating sharing of public charge points by untrusting agents.

[27] Mode 3 charging is AC charging from 2.3kW–22kW

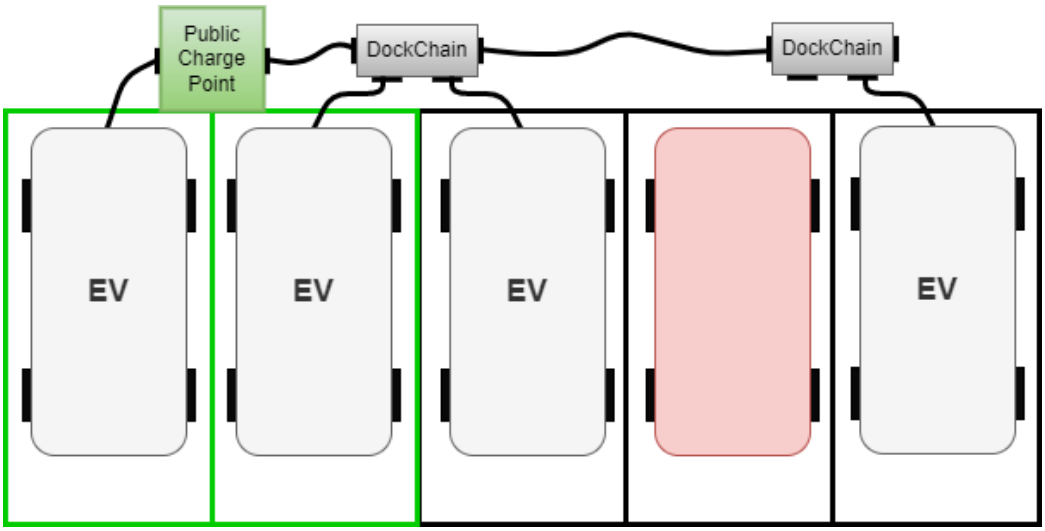


Figure A.1: Two DockChains chained together, extending the reach of the right hand side outlet on a public charge point. Three EVs have connected to a single outlet via the DockChains, and one potential extra charging socket has been blocked by a regular vehicle.

The first prototype of DockChain compatible with mode 4[28] charge points has also been developed during this PhD, although these innovations are the primary focus of Go Eve[29], a company co-founded by the present author focussed on commercialising DockChain technology. As such, the specifics of the DC implementation are beyond the scope of this thesis. The scheduling and DLT-based trading approaches presented here are equally applicable to DC rapid charging, but we focus our attention in this appendix on the AC system presented in [103]. The interested reader can visit the Go Eve website for up-to-date information on commercially available DockChain solutions. Figure A.2 shows a render of DockChain deployed in a car park which has been used for marketing purposes by Go Eve. Figure A.3 shows further marketing renders of Go Eve's Dockchain products.

The appendix is organised as follows. In Section A.1 we give a high level overview of the main components of the DockChain system (we focus on mode 3 charging as presented in [103]) and describe its operation. In Section A.2 we outline a framework for cooperative charging based on earliest deadline first (EDF) scheduling and discuss V2V and V2G trading in this context. We present a typical use case of this system and provide simulations in Section A.3. We summarise results and suggest directions for future research and development in Section A.4.

A.1 System Overview

The proposed solution to the problem of rapidly expanding EV charging networks is a lightweight and low cost device that can be connected to existing public charge points and can sequentially reroute the charge point's power to multiple EVs. A prototype is pictured in Figure A.4. This embodiment of DockChain has two outputs for EVs on each device and an additional socket for connecting additional DockChains to form a chain, as illustrated in Figure A.1. A similar device was presented in [108] for the purpose of monetising private charge points in homes or business. This concept was then extended in [109] with the idea of cascading these devices to allow sharing of public charge points, but the means of accessing the power of public charge points was not addressed. The DockChain system presented here utilises the protocols by which EVs must interact with public charge points and vice versa as outlined in IEC-61851 standards [110].

[28] Mode 4 charging is DC rapid charging 22kW–350kW

[29] Go Eve have been recipient of numerous innovation awards including University College Dublin's start up of the year and a Department of Transport grant. Visit `https://www.goeve.co.uk/about` for more information.

Figure A.2: A render of DockChain devices deployed in a car park. Produced for marketing purposes by Go Eve.

Figure A.3: Renders of DockChain products available from Go Eve. Left: a DC charging cabinet used to a supply a chain of DockChain devices. Centre and right: DockChain options for different deployment locations. Produced for marketing purposes by Go Eve.

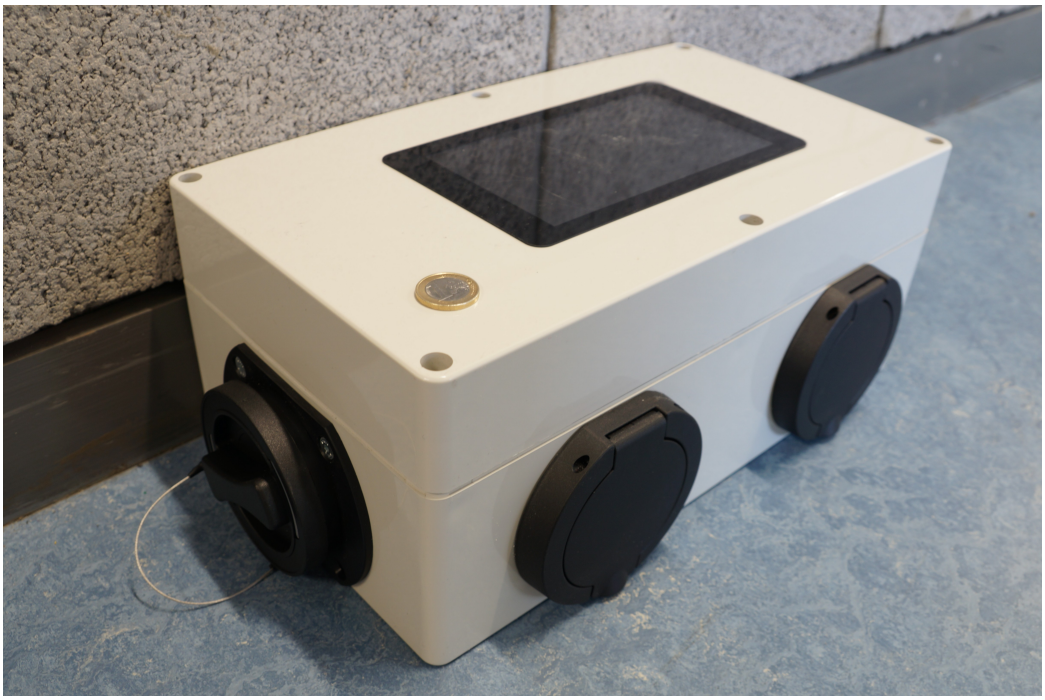


Figure A.4: A prototype DockChain device.

A.1.1 Hardware Layer

IEC-61851 compliant Electric Vehicle Supply Equipment (EVSE) communicates with EVs over the Control Pilot (CP) on a standard Type 2 plug. The EVSE produces a ±12 V PWM signal, the duty cycle of which indicates the charge rate available from the EVSE. On the EV side, a pull-down resistor indicates the state of the EV to the EVSE i.e. its readiness to charge. What we would like to be able to do when we plug in a DockChain device to the EVSE is to have the DockChain appear identical to an EV from the perspective of the EVSE. Meanwhile we would like the DockChain to appear identical to an IEC-61851 compliant EVSE from the perspective of the EV. This *man-in-the-middle* approach allows us to take control of the power available from the EVSE and distribute it as we please to EVs in compliance with IEC-61851 standards. Figure A.5 illustrates how DockChain handles the CP signal. The inlet coming from the EVSE passes in to an array of resistors which we can control to mimic different EV states via the Raspberry Pi 3B+ controller, as indicated in the figure. The CP signal (whose voltage has now been pulled down by these resistors) then passes through comparator circuits to produce two new ±12 V signals identical to the original signal produced by the EVSE. One of the generated CP signals is directed to an outlet intended for connecting further DockChain devices and the other can be directed to either of the outlets available for connecting EVs.

The approach described above and illustrated in Figure A.5 allows multiple DockChain devices to be connected in series—the CP signal from the output socket for additional DockChain devices will be identical to that produced by the EVSE. Note that bidirectional charging will require different hardware due to the more complex communication protocols required and the more complex routing of power, whether it be V2V or V2G.

A.1.2 Network Layer

In order to have multiple DockChain devices connected to a single charge point and have the available power reliably distributed, we must enable communication both between adjacent DockChain devices and between users and DockChain devices. Bluetooth communication between adjacent DockChain devices allows them to coordinate charging and ensure that the overall power demand on the EVSE never exceeds the capacity indicated by its CP signal. As for users communicating

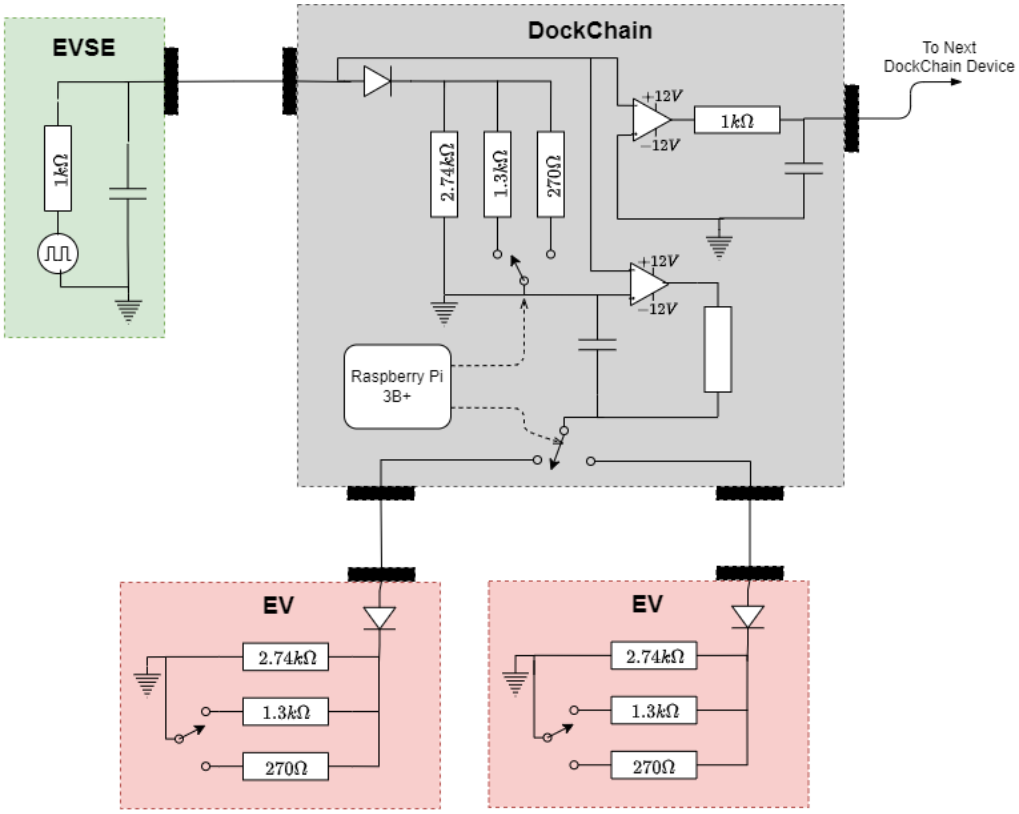


Figure A.5: The DockChain manipulates the CP signal so as to appear like an EV from the viewpoint of the EVSE and appear like an EVSE from the viewpoint of an EV or another DockChain.

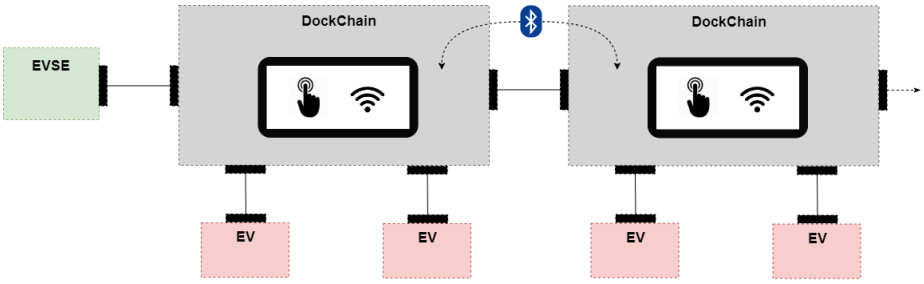


Figure A.6: A chain of DockChain devices: adjacent DockChain devices communicate over Bluetooth, and EV owners communicate with DockChain devices via a touchscreen interface and/or a smartphone app.

with the system to indicate their charging needs and the system communicating its status to users, we can achieve this in two ways: the touchscreen interface allows two way communication directly between the users and the system; a smartphone application presents updates on charging status to the users in real-time and allows modification of desired charge and deadlines etc. by users.

A.1.3 DLT-Based Trading Layer

The DLT-based trading layer facilitates two kinds of transaction. Firstly, we would like to enable EVs to easily and securely make real-time payments to the DockChains for the charging service. The cost per unit of electricity could, for instance, depend on grid capacity and renewable generation at any given time. Secondly, we would like to enable V2V trading. When a number of DockChains have been deployed at a charge point, multiple EVs will be able to contend for the available power. DockChain is capable of intellegent scheduling of charging which takes each user's requirements into account. This is discussed in further detail in Section A.2, but with an algorithm for optimally scheduling charging, we would like users to be able to trade places in the queue by making direct payments to other EVs ahead of them. Direct V2V payments can also be made for V2V charging with the addition of bidirectional charging hardware.

A.2 Cooperative Charging Frameworks

Various cooperative charging policies have been discussed in relation to the prior iterations of this charge point extender [108, 109]. We now seek to formalise the problem and propose a means of implementing such policies.

We wish to distribute the total capacity P of an EVSE among n EVs that are connected to the EVSE via a chain of DockChain boxes. In what follows, we assume that a chain of DockChain boxes acts as a single central controller which may be achieved by configuring the box connected directly to the EVSE as the master. The algorithms described in this section may be implemented in a distributed manner across the boxes, however we do not discuss this further here. The controller of the master box has access to the following information on the EVs connected to the system:

- $x(t) \in \mathbb{R}^{n\times 1}$, the state of charge (SoC) of the EVs at each socket at time t;
- $x_{ref} \in \mathbb{R}^{n\times 1}$, the desired SoC for each EV;
- T_i, the deadline for charging the EV at socket i;
- $\rho_i(t) \in [0, 1]$, the charging efficiency of EV at socket i at time t: $\rho(t) = \text{diag}\,(\rho_i(t))$;
- $u(t) \in \mathbb{R}^{n\times 1}$, the binary control vector indicating the state (on/off) of each socket at time t.

We now present an algorithm for choosing $u(t)$.

A.2.1 Earliest Deadline First Scheduling

When an EV connects to the system, the owner can request their desired final state of charge, $x_{ref,i}$, and their deadline, T_i. We aim to assign a charging slot to this EV to guarantee that it will be ready for the requested deadline, provided the request is feasible under the current conditions, including the state of other vehicles connected and the charging rate of the EVSE, P. We can borrow an important result from CPU task scheduling [111] which tells us that by catering to the task with the earliest deadline first (EDF), we make optimal use of the resource in this case because we know the deadlines and the time required for each task ahead of time. Note that this is not quite like a typical CPU task scheduling problem because tasks are not periodic. Nonetheless, an implementation of an EDF algorithm for this system is described in Algorithm 13.

Algorithm 13 Earliest Deadline First Scheduling

Initialise:
1: $\pi \leftarrow$ empty list
Repeat:
2: $t \leftarrow$ current time
3: **if** event at socket i **then**
4: $(\pi, \text{status}_i) \leftarrow$ UpdateSchedule(i, π)
5: **if** π is empty **then**
6: $u(t) = \mathbf{0}$ ▷ switch off all sockets
7: **else**
8: $i^* \leftarrow$ index of socket listed first in π
9: $u(t) = \mathbf{e_{i^*}}$ ▷ switch on socket i^*
10: **end if**
11: **end if**

12: **function** UpdateSchedule(i, π_{old})
13: **if** EV present at socket i and $x_{ref,i}(t) > x_i(t)$ **then**
14: $\pi_{new} \leftarrow \pi_{old}$
15: $\tau_i \leftarrow (x_{ref,i}(t) - x_i(t))/P$
16: π_{new}.append$(\{\tau_i, T_i\})$
17: sort π_{new} by deadline ▷ earliest first
18: $l \leftarrow$ index of last socket in π_{new}
19: **if** $t + \sum_{i=1}^{n} \tau_i \leq T_l$ **then** ▷ schedulable
20: **return** $(\pi_{new}, \text{success})$
21: **else** ▷ not schedulable
22: **return** $(\pi_{old}, \text{failure})$
23: **end if**
24: **else**
25: $\pi_{new} \leftarrow \pi_{old}$
26: π_{new}.remove$(\{\tau_i, T_i\})$
27: **return** $(\pi_{new}, \text{success})$
28: **end if**
29: **end function**

Note that schedule updates in Algorithm 13 are triggered by events at the sockets. Such events include:

- a new EV is connected at socket i;
- the EV at socket i is disconnected;
- the desired battery level, $x_{ref,i}(t)$, or the deadline, T_i are updated for socket i;
- the EV at socket i has reached its desired battery level $x_{ref,i}(t)$.

The status of socket i, referenced in Line 4 of Algorithm 13, indicates whether a requested battery level and deadline is schedulable for the EV at socket i or not.

A.2.2 V2V Trading

If the status is "failure", then the request is not schedulable and the EV owner has two options: they can request a lower battery level or a later deadline; or they can request to trade places in the queue with other EV owners who connected before them in exchange for a payment.

A.3 Case Study: City Centre Workplace

Consider the following illustrative deployment scenario, dimensioned using Irish commuter data and typical EV battery usage: a company has installed a 7kW EVSE at a workplace car park in a city centre location, but wishes to expand the number of sockets to facilitate the rising number of EV owners in the company.

A.3.1 Dimensioning

Between 8am and 6pm we have a capacity of around 70kWh to charge EVs from each socket of the EVSE. Assuming the average city commute distance to be around 10km each way [30] an inefficient EV would only need around 4kWh (0.2kWh/km) of charge to cover their commute
The 2018 Nissan leaf claims a range of 270km for the 40kWh battery i.e. using less than 0.15kWh/km, and this can be improved upon with city driving. This means that in the use case of interest here (day-to-day commuting within a city) this 7kW EVSE should reasonably have the capacity to service 17–18 EVs for their full commuting needs. We need to unlock this capacity by allowing more EVs to connect and by managing the distribution of the available power so that everyone gets their fair share.

For this use case, we could deploy 8 DockChains at each EVSE socket, allowing 16 average commuters to charge more than they would need for their daily commute from a single socket.

A.3.2 Implementation

Let us now provide preliminary results for each of the Algorithms described in Section A.2 for a typical workplace deployment. For the purpose of a clear illustration with a legible plot, we simulate just 4 EVs and scale their expected charging requirements, calculated above, by a factor of 4 to be representative of a similar system with 16 EVs.

Standard Use

Consider a typical work day in which all users arrive in the morning at similar times and leave in the evening at similar times. The first section of Table A.1 contains the charge and timing requirements of each EV in this scenario. We can clearly see from the results in Figure A.7 that the EDF algorithm successfully schedules all charging without issue, as expected.

Changing Deadline

Suppose now that at 9:30am, the owner of EV 4 has realised they need their car by 1pm after all as they are taking a half-day. They update their charging deadline to reflect this, as the schedule is still feasible. The second section of Table A.1 contains the updated charge and timing requirements.

We see in the results presented in Figure A.8 that at 9:30am, when EV 4 updates their deadline, the algorithm updates the schedule to ensure that EV 4 charges on time, and all other EVs still charge on time.

V2V Trading

Suppose now that, in addition to the changes made by EV 4 above, the owner of EV 2 has realised during the day that they will actually need around 30 kWh in their battery for a long journey that evening. The maximum charge achievable under the conditions at this stage is around 20 kWh.

[30] The average straight line distance for commuters in Dublin City is actually only 5.88km according to the 2016 census of Ireland [112]

EV	Initial SoC	Arrival Time	Desired SoC	Deadline
	1) Standard Use			
1	6 kWh	8:10am	27 kWh	5pm
2	4 kWh	8:20am	12 kWh	5:30pm
3	10 kWh	8:30am	18 kWh	4:40pm
4	5 kWh	8:50am	25 kWh	6pm
	2) Changing Deadline			
1	6 kWh	8:10am	27 kWh	5pm
2	4 kWh	8:20am	12 kWh	5:30pm
3	10 kWh	8:30am	18 kWh	4:40pm
4	5 kWh	8:50am	25 kWh	**1pm**
	3) V2V Trading			
1	6 kWh	8:10am	**17 kWh**	5pm
2	4 kWh	8:20am	**30 kWh**	5:30pm
3	10 kWh	8:30am	18 kWh	4:40pm
4	5 kWh	8:50am	25 kWh	1pm

Table A.1: Scenarios 1, 2, and 3. Changes from the previous scenario are highlighted in bold.

The owner of EV 2 broadcasts to the other EVs in the system that they would like to pay someone for their charging slot. EV 1 accepts the offer and EV 2 makes a payment in DLT tokens to the digital wallet of EV 1, and in return, the requested charge of EV 1 is reduced by 10 kWh to allow EV 2 get the full 30 kWh desired. The third section of Table A.1 contains the charge and timing requirements of each EV in this scenario.

The results, presented in Figure A.9 for this scenario, show that the EDF algorithm adapts successfully to this change in desired charge.

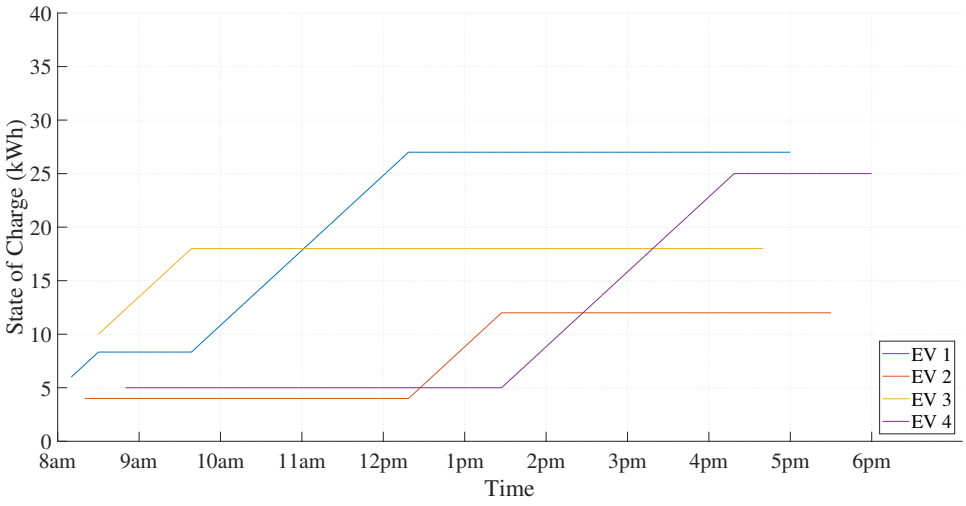


Figure A.7: Earliest deadline first algorithm: standard use.

A.4 Summary

The system described here can be rapidly deployed at existing EV charge points at a low cost compared with installation of additional charge points. The system allows multiple EVs to connect to a single charge point, and an adaptive scheduling algorithm ensures that the needs of each user are met, where possible within the constraints of the system. The DLT-based trading layer allows users to trade places in the queue for monetary reward. Further extensions of this system

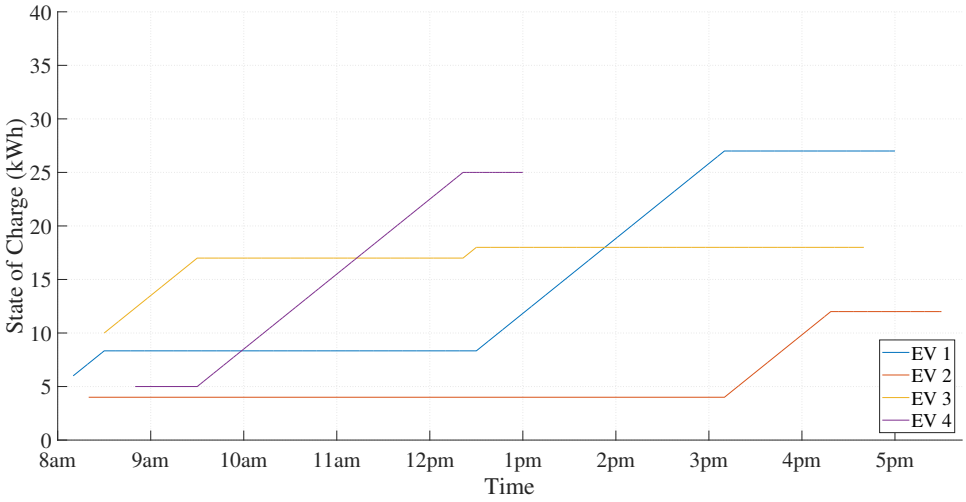


Figure A.8: Earliest deadline first algorithm: changing deadline.

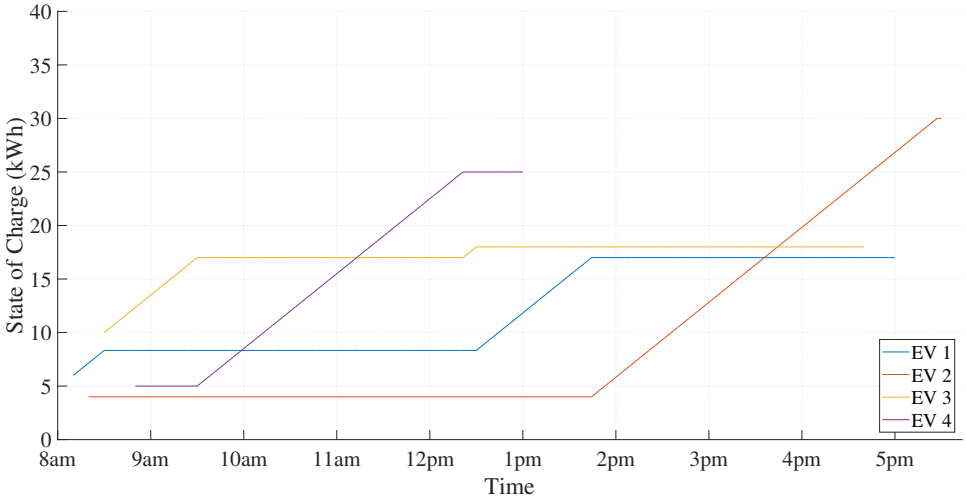


Figure A.9: Earliest deadline first algorithm: trading places in the queue.

could incorporate V2V and V2G charging, and the DLT-based trading layer could also seamlessly facilitate trading of charge in this way. These extensions would allow the system to offer a valuable service to the grid in providing a readily available pool of EVs with the capacity to sell stored energy and allow EV owners to monetise their vehicle's battery when it is not in use.

Appendix B

SPToken: DLT-augmented Reinforcement Learning

***Abstract*— Recommender systems now play a role in delivering personalised services to millions of people on a daily basis. The providers of these services must, by the very nature of recommender systems, gather personal data on their users to provide updated recommendations. However, the manner in which data has been stored and used by organisations in recent years has raised ethical concerns and created a demand for more decentralised and transparent modes of data management. The proliferation of distributed ledger technology presents a new paradigm for data management which is more private and transparent than the current model, and returns control over data to consumers and regulatory agencies alike. In this appendix, we present a framework for building recommender systems in which data is sampled and managed on a distributed ledger. We apply this framework to a route recommendation problem using reinforcement learning and show that our approach not only implements a more ethical model for data management but supports efficient sampling and faster learning. This is joint work with Roman Overko of University College Dublin, Dr. Rodrigo Ordónez-Hurtado and Dr. Sergiy Zhuk of IBM Research, Ireland, and Dr. Pietro Ferraro and Prof. Robert Shorten of Imperial College London. This research also appears in [52].**

Companies such as Facebook, Google, Amazon, Waze and Garmin are just some examples of the many corporations that have built successful service delivery platforms using personalised data to develop recommender systems. While products gleaned from data mining of personal information have undoubtedly delivered great societal value, they also have given rise to a number of ethical questions that are causing a fundamental revision of how data is collected and managed. Among the most pressing ethical issues are the following:

1. preservation of individuals' privacy (including GDPR compliance);
2. the ability for individuals to retain ownership of their own data;
3. the ability for consumers and regulatory agencies to confirm the origin, veracity, and legal ownership of data, products and services;
4. protection against misuse by malevolent actors.

It is in these context that DLT has much to offer and our objective in this appendix is to design a DLT-based architecture to address these concerns. We are particularly interested in using

DLT to realise crowdsourced collaborative recommender systems to support a range of mobility applications for smart cities. The DAG-based distributed ledgers which have been the primary focus of this thesis are particularly suitable for such applications for a number of reasons. First, we require the ledger to facilitate high-frequency micro-transactions in order to support the rapid exchange of information between the multitude of IoT devices found in cities. Second, as the DLT must support multiple control actions and recommendations in real-time, transaction times should be fast with low or zero transaction fees. Finally, the DLT should penalise malevolent actors who attempt to spam the system or lie to attack the design of any recommender system based on the DLT.

It should be noted that wrapping a DLT layer around personal information will fundamentally change the business model of many companies. Many corporations currently monetise recorded personal data with no explicit reward returned to the owner of such data (other than personalised recommendations or free access to products in return for the collected data). If such data is no longer available free of charge to these corporations, that will surely jeopardise existing business models. In future, most data will be privately held and not available in a public manner, and companies seeking to develop services will need to purchase this data to sample an unknown density. In this context, a fundamental challenge is to do this at minimum cost, as quickly as possible, given some desired level of accuracy (e.g., a minimum quality of service) and to develop a set of tools to enable such companies to sample these large data sets, secured in a distributed ledger, in an economic manner.

A second challenge arises from the design of the recommender systems themselves. In many important applications, the development of complex decision making tools is inhibited by difficulties in interpreting large-scale, aggregated data sets. This difficulty stems from the fact that data sets often represent *closed-loop* situations, where actions taken under the influence of decision support tools (i.e. recommenders), or even due to probing of the environment as a part of the model building, affect the environment and consequently the model building itself. Recently a number of papers have appeared highlighting the problem of recommender design in closed loop environments [113, 114, 115, 116, 117, 118]. Even in cases when there is a separation between the effect of a recommender and its environment, the problem of recommender design is complex in many real world settings due to the challenge of sampling and obtaining real-time data at low cost.

In this appendix we bring both of the above problems together in one framework. In particular, we consider the problem of sampling an unknown density representing traffic flow in a city, using a DLT-based architecture that allows for data collection through secured access points, and without perturbing the density through probing actions. Specifically, we will use reinforcement learning (RL) [84, 119] to sample the density in order to build a model of the environment. However, classical implementations of RL are usually not applicable in many smart city applications due to their long training time, the disruptive effects of probing, and poor availability of data. We shall demonstrate how DLT allows us to achieve rapid probing actions without affecting the environment, and also enabling individuals to retain ownership of their own data while being rewarded for contributing to the RL algorithm.

B.1 Related Work

Our work brings together ideas from many areas. The first key area, DLT, has been the primary focus of this thesis. The interested reader can refer Chapter 1 and the references therein for an overview of DLT. For the purpose of this application, we are mostly interested in DAG-based distributed ledgers due to the fact that these architecture are designed to facilitate high-frequency microtrading, they place a low computational and energy burden on devices, they do not require transaction fees, and transactions are pseudo-anonymous [34]. In terms of mobility applications, we note that several blockchain-based DLT architectures have already been proposed. Recent examples include [120, 121, 122] and the references therein. To the best of our knowledge, our work is the first to use a DAG-based distributed ledger to support distributed machine learning (ML) algorithms.

In terms of ML, we borrow heavily from RL, Markov decison processes (MDPs), and, in particular, crowdsourced ML. The literature on MDPs and RL algorithms is vast and we simply point the reader to some relevant publications in which some of this work is discussed (see [117, 123, 124]). With specific regard to RL and mobility, some applications are presented in [125, 126, 127, 128, 129]. We exploit the idea of using crowdsourced behavioural experience to augment the training of ML algorithms (see a recent survey for an overview of this area in [130]).

Finally, it is worth mentioning that we are ultimately interested in the design of recommender systems that account for feedback effects in smart city applications. In [131, 132], different information is sent to different agents in an attempt to mitigate closed-loop effects. An alternative and more formal approach is presented in [117]. There, the authors attempt the identification of a smart city system from closed-loop data sets. Similar issues have drawn interest from various domains including economics [133], recommender systems [134, 114], physiology [135], and control engineering in the context of smart cities [115].

B.2 SPToken: DLT for Crowdsourced Smart Mobility

Our aim is to design a DLT-based system for crowdsourcing data in a smart mobility environment. In particular, we explore how to apply this framework to an RL setting where a third party is interested in acquiring information from vehicles in order to solve an optimisation problem.

The underlying idea is to use a set of virtual *tokens* as a proxy to indicate specific geographical points of interest whose states and conditions (e.g., position, speed, nearby air pollution) are of significance for dedicated algorithms. In routing algorithms, for example, we are interested in maximising the expected reward (relative to an objective function) for taking a specific route across a city. To make this process clearer, consider the following example. Figure B.1 shows an instance of a typical scenario where two road junctions A and B are connected to one another through the road segment $\overrightarrow{AB}$. At time T_A, a vehicle updates the ledger with some collected information x^{T_A} (e.g., air pollution level, travel time) by registering at a given visited intersection (A, in this example). Intuitively, this can be depicted as if the vehicle leaves a token κ with associated information x^{T_A} at junction A. Then, a new vehicle passing via junction A and directed to junction B can "collect" token κ and, as it passes by junction B at time $T_B > T_A$, it updates the ledger with new information x^{T_B} regarding route link $\overrightarrow{AB}$ and the new position of the token. It is noteworthy that in Figure B.1 a vehicle leaves the token when it deviates from the token route. Thus, a new car that passes by junction B whose immediate future trajectory coincides with the token's route will be able to collect the token and the procedure is repeated for a new road segment.

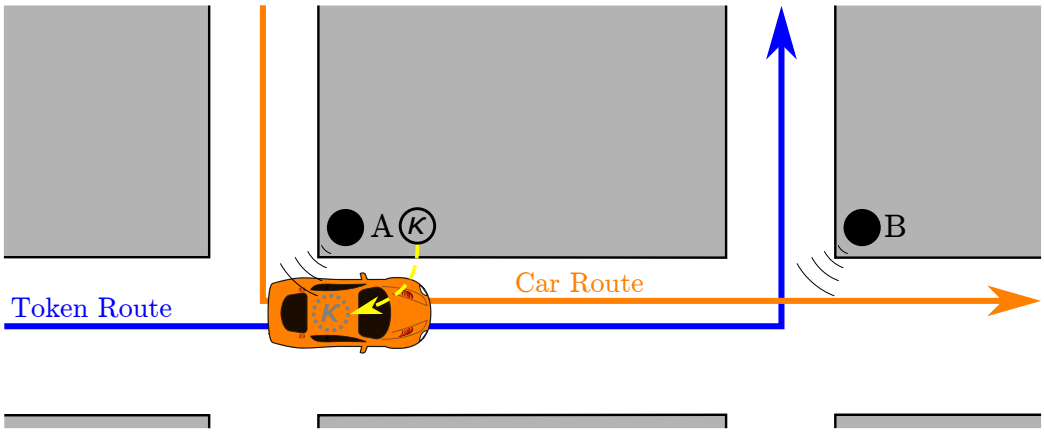
The concept of using tokens to be deposited at specific locations where measurements are needed perfectly conforms with a DLT-based system. In fact, it is natural to use distributed ledger transactions to update the position of available tokens and register the associated data to the points of interest by using transactions (which can be done, for example, using smart sensors at various junctions linked to digital wallets, as shown in Figure B.1). Of course, the design of such a network poses a number of challenges that need to be addressed.

- *Privacy:* In the DLT, transactions are pseudo-anonymous[31]. This is due to the cryptographic nature of the addressing, which is less revealing than other forms of digital payments that are uniquely associated with an individual [31]. Thus, from a privacy perspective, the use of DLT is desirable in a smart mobility scenario.

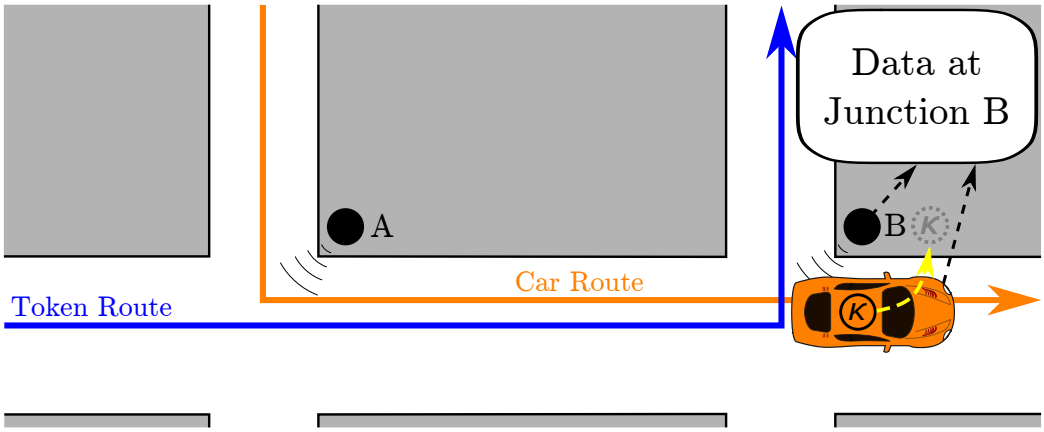
- *Ownership:* Transactions in the DLT can be encrypted by the issuer, thus allowing every agent to maintain ownership of their own data. In the aforementioned setting, the only information required to remain public is the current ownership of the tokens.

- *Microtransactions:* Due to the amount of vehicles in the city environment, and also due to

[31] `https://laurencetennant.com/papers/anonymity-iota.pdf`



(a) A vehicle passes through a junction A where another car has recently issued some data (this is displayed by a token). This makes the agent eligible to write transactions to the ledger.



(b) The same vehicle passes through junction B. It then writes some data, relative to the road link $\overrightarrow{AB}$, to the ledger and deposits the token so that another vehicle will be able to collect it.

Figure B.1: The sequence to issue new data from vehicles. Here κ denotes a token.

the need of linking the information to real-time conditions (such as traffic or pollution levels), there is the demand for a fast and large data throughput.

- *Resilience to Misuse:* The system must be resilient to attacks and misuse from malevolent actors. Typical examples include double spending attacks and spamming of the system as discussed in earlier chapters.

To meet all the design objectives described above, in the next section we propose *Spatial Positioning Token* (SPToken), a platform built on top of a suitable distributed ledger with an additional regulatory policy in order to prevent agents from adding transactions that do not possess any relevant data. Specifically, SPToken makes use of PoP to authenticate transactions: for a transaction to be authenticated, it has to carry proof that the agent was indeed in an area where a token was present, which is achieved via special nodes called *observers* linked to physical sensors in a city[32]. Whenever a participating car passes by an observer that is in possession of a token, a short range wireless connection is established (e.g., via Bluetooth) and the token is transferred to the vehicle's account if the requirements are met (e.g., the immediate vehicle's and token's future trajectories intersect). To deposit the token and to issue a transaction containing data, the agent

[32] A sensor can be a fixed piece of infrastructure, or a vehicle whose position is verified.

needs to pass by another observer and establish a short range connection. See Figure B.1 for a better understanding of this process. This process ensures that vehicles have to be physically at the observation points to be able to issue transactions.

B.3 Reinforcement Learning with SPToken

Our objective now is to implement an RL strategy using the token-passing architecture described in the previous section. Specifically, instead of using vehicles as RL agents [118] to probe an unknown environment, we use tokens "jumping" among vehicles to effectively create virtual agents and emulate the behaviour of commanded agents designed to probe the surroundings of arbitrary routes. For this, we employ a modified version of the recently proposed RL algorithm called **U**pper **B**ounding the **E**xpected Next State **V**alue (UBEV) [136]. UBEV involves a combination of *backward induction* with maximum likelihood estimation to (i) construct optimistic empirical estimates of state transition probabilities, (ii) assign empirical immediate reward, and (iii) compute optimal policy. In fact, our design of the state-action space allows us to avoid estimating the transition probabilities, which significantly reduces the training time. Effectively, the algorithm learns only the reward function which describes the environment (e.g., traffic patterns in a city).

Since the training time is a common disadvantage of RL algorithms, we propose to launch a high number of independent tokens, which act as virtual vehicles and use the same MDP's policy matrix to explore different areas of a city. Further details of the proposed approach, together with the corresponding experimental assessment, are provided in the following sections. In particular, we experimentally assess:

- how fast the system learns to avoid traffic jams;
- how quickly the system returns to the shortest path policy once the traffic jams clear up;
- how the training time varies with respect to the number of independent tokens.

The original UBEV algorithm in [136] performs a standard expectation-maximisation trick. Namely, it first fixes the state transition probabilities of the MDP and the expected reward estimates, and uses backward induction to design the optimal deterministic policy which maximises the expected reward. Next, this policy is used to probe the environment, and the statistics collected over the course of probing are used to update transition probabilities by employing a standard "frequentist" maximum likelihood estimator [137], which simply computes the frequencies of transitioning from one state to another subject to the current action (that can be a function of the current state). Then, the optimal policy (for the updated estimates of the transition probabilities and reward) is recomputed again. This procedure is treated as an *episode* of the training process and is iterated until convergence (as demonstrated in [136]).

B.3.1 Modified UBEV algorithm

Our decision problem is a finite horizon problem with time horizon length H, where we assume that the model is known and that the environment is fully observable. An MDP can be represented as a tuple $\langle \mathcal{S}, \mathcal{A}_s, \mathbf{P}, \mathcal{R} \rangle$, where

- $\mathcal{S}$ is the set of states, with $|\mathcal{S}| = S$ being the number of states;
- $\mathcal{A}_s$ is the set of allowable actions, with $|\mathcal{A}_s| = A_s$ being the number of allowable actions in state s, $\mathcal{A} = \bigcup_{s \in \mathcal{S}} \mathcal{A}_s$, and with $|\mathcal{A}| = A$ being the total number of actions;
- $\mathbf{P}(s'|s, a, t)$ is the probability of transition from state s under action $a \in \mathcal{A}_s$ to state s' at decision epoch $t \in \mathcal{H}$, $\mathcal{H} = \{1, 2, \ldots, H\}$;
- $\mathcal{R}(s, a, t)$ is the reward of choosing the action $a \in \mathcal{A}_s$ in the state s at decision epoch $t \in \mathcal{H}$.

In an MDP, an agent (i.e., the decision maker) chooses action $a_t \in \mathcal{A}_s$ at time $t \in \mathcal{H}$ based on observing state s_t, and then receives a reward r_t. The trajectory of the MDP is defined as follows: it is assumed that $s_{t+1} \sim \mathbf{P}(\cdot|s_t, a_t, t)$, i.e., the state at time $t+1$ is drawn from a distribution $\mathbf{P}$ which depends on s_t, $a_t \in \mathcal{A}_s$ and decision epoch t. In this case, the total expected reward associated to the policy $\pi : \mathcal{S} \to \mathcal{A}$ is defined as

$$\rho(\pi) \coloneqq \mathbb{E}_{s_1 \dots s_H}\left[\sum_{t \in \mathcal{H}} \mathcal{R}(s_t, \pi(s_t, t), t)\right] = \sum_{s \in \mathcal{S}} \mathbf{P}_0(s) V_1^{\pi}(s)\,, \tag{B.1}$$

where $\mathbf{P}_0$ is the distribution of the initial state, and V_t^{π} is the value function from decision epoch t for policy π, formally defined as follows:

$$\begin{aligned} V_t^{\pi}(s) &= \mathcal{R}(s, \pi(s,t), t) + \sum_{s' \in \mathcal{S}} \mathbf{P}(s'|s, \pi(s,t), t) V_{t+1}^{\pi}(s'), \\ V_{H+1}^{\pi} &\coloneqq 0. \end{aligned} \tag{B.2}$$

Then, the goal of an agent is to find an optimal trajectory which maximises the expected reward (B.1), and the optimal MDP policy (i.e., the policy maximising (B.1)) is calculated through the backward induction process given by:

$$\begin{aligned} \pi(s,t) &= \arg\max_{a \in \mathcal{A}_s} \left\{ \mathcal{R}(s,a,t) + \sum_{s' \in \mathcal{S}} \mathbf{P}(s'|s,a,t) V_{t+1}^{\pi}(s') \right\} \\ \pi(s,H) &= \arg\max_{a \in \mathcal{A}_s} \mathcal{R}(s,a,H). \end{aligned} \tag{B.3}$$

We are now in a position to present the **M**odified **UBEV** (MUBEV) algorithm as described in Algorithm 14 which is a result of adapting the original UBEV algorithm to our target problem. Our first modification is to use a specific type of state-action space. A state is represented as a collection of road links, while the action space consists of possible directions of connections between the edges of a road network[33]. Namely, we apply the actions '**s**'—go straight, '**l**'—turn left, '**L**'—turn partially left, '**R**'—turn partially right, '**r**'—turn right, and action '**u**'—stay in the same state (which prevents leaving the destination state). We also exclude U-turns to favor the exploration of the environment, as U-turns may result in undesirable recurrent attempts to use the shortest path policy. The proposed model of the state-action space, based on directions between road links, allow us to provide the algorithm with the set of predefined trivial transition probabilities.

For example, let us construct stochastic rows of transition matrices for all possible transitions from state s_0 assuming that actions '**s**', '**l**', and '**r**' are allowable in this state as shown in Figure B.2. Clearly, it is not required to learn such trivial transition probabilities, which is a significant advantage especially for large road networks. Note that, in our model, the action '**u**' (stay in the same state) is allowable at each state $s \in \mathcal{S}$.

Our second modification addresses the following situation. At the beginning of the training, there is little or no information on the reward distribution, and the algorithm explores rather than exploiting. By default, the original algorithm always selects the first component of Q if all the components of the Q-function are equal(Algorithm 14, line 24), and thus it probes the environment without any preference in terms of the direction of the exploration. In contrast, we force it to stick to the shortest path policy whenever $Q_i = Q_j$, $\forall i, j$, so that our algorithm explores the area around the shortest route. If the agent faces a traffic jam after a certain action $a \in \mathcal{A}_s$, it gets delayed, which in turn introduces the negative reward for the action $a \in \mathcal{A}_s$ at state s. As a result, the reward distribution changes, and the shortest path policy is amended to avoid the jam by looking for a detour. By operating in this fashion, we sample along near optimal trajectories.

[33]In this work, we do not consider lane-changing behaviour for the agents on multi-lane roads.

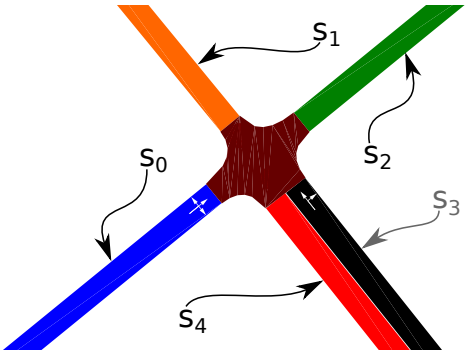


Figure B.2: A piece of a road network with five road links representing states s_0, s_1, s_2, s_3, s_4. Note that state s_3 is marked in gray since it is not accessible from state s_0.

Concerning the third modification, we aim to launch multiple tokens always starting at different (randomly sampled) origins and having the same destination. All these tokens follow the same MDP's policy matrix, and the corresponding collected statistics are then used to update the policy. Therefore, learning and adaptation happen more rapidly.

Finally, we propose a stationary model of the MDP (i.e., transition probabilities and the reward distribution do not vary with time) where each independent agent (token) contributes to the MDP's reward matrix, and they all use new updated policy in the next episode of the learning process.

B.3.2 Notation for MUBEV and the Reward Function.

In Algorithm 14: $\mathcal{S}$ is the set of states; $\mathcal{A}_s$ is the set of allowable actions in state s, $\mathcal{A} = \bigcup_{s\in\mathcal{S}} \mathcal{A}_s$; S and A denote cardinality of finite sets $\mathcal{S}$ and $\mathcal{A}$ respectively; A_s denote cardinality of a finite set $\mathcal{A}_s$; H is the length of the MDP's time horizon, with $\mathcal{H} = \{1, 2, \cdots, H\}$; $\mathbf{P}$ is an array of predefined transition probabilities; Π_{SP} is the shortest path policy; M is the number of MUBEV tokens; δ is the failure probability (see [136] for details); $n(s, a, t)$ is the number of actions $a \in A_s$ taken from state s at time t; $R(s, a, t)$ is accumulated reward from state s under action $a \in \mathcal{A}_s$ at time t; $\hat{V}(s, t')$ is the value function from time step t' for state s; $\hat{Q}(s, a, t)$ is the Q-function for the appropriate state, action and time [136]. Initial values of elements in arrays n, R, $\hat{V}$ and $\hat{Q}$ are zeros for all $s \in \mathcal{S}, a \in \mathcal{A}_s, t \in \mathcal{H}, t' \in \{1, 2, \cdots, H+1\}$. Additionally: r_{max} is the maximum reward that the agent can receive per one transition; V_{max} is the maximum value for next states; $\hat{V}(\cdot, t+1)$ and $P(\cdot, s, a, t)$ denote vectors of length S, and $\hat{Q}(s, \cdot, t)$ is interpreted as a vector of length A_s; ϕ is the width of the confidence bound [136]; e is the Euler's number; $\hat{r}(s, a, t)$ is normalised reward from state s under action $a \in A_s$ at time t; and r and EV are auxiliary variables. Vector $\tilde{s}$ is a vector of initial states of MUBEV tokens, which is uniformly sampled in range from 1 to S with no repeated entries. The agents (tokens) interact with the environment each time step $t \in \mathcal{H}$, and receive reward r_t determined by the reward function defined in Function 1.

Concerning Function 1, it returns total reward, i.e., distance reward plus time reward, at time t. Additionally: $\tau(s_{t+1})$ is actual travel time on edges that correspond to state s_{t+1}; α is a scale factor that increases minimum travel time on edges due to traffic uncertainties; β is a parameter used for faster learning of congestions; ω_D and ω_T are the weights of distance and time reward, respectively; Ω is the absolute value of penalty given to the agent if it takes action **'u'** if not at the destination state or when it leaves the destination; $D(s_t)$ is the shortest route length from state s_t to the destination state s_f; and $L(s_t)$ is the length of edges that correspond to state s_t. Finally, $RY(s_{t+1})$ is the duration of yellow and red phases of traffic light signals (TLS) that control edges that represent state s_{t+1}; if all edges in some state are not controlled by a TLS, we apply $RY = 0$ for that state. If some edges are not controlled by traffic light signals, we employ the edge coefficient EC for them (Function 1, line 15) which is computed as follows: if the length L_{t+1} of edges that correspond to state s_{t+1} is smaller than the average length of edges included in states

$\bar{L}$, then $EC(s_{t+1}) = \left[L(s_{t+1})/\bar{L}\right]^4$, otherwise $EC = 1$.

B.4 Simulations

In the following application, we are interested in designing a recommender system for a community of road users. We distribute a set of MUBEV tokens so that the uncertain environment can be probed by passing these tokens from vehicle to vehicle via the DLT. The token positioning is determined by the operation of the MUBEV algorithm and vehicles possessing a token are permitted to write data to the DLT. We refer to such vehicles as virtual MUBEV vehicles. In this way, the token passing emulates the behaviour of real agents (vehicles) that are probing the environment.

For the experimental evaluation of our proposed approach, we designed a number of numerical experiments based on traffic scenarios implemented with the open source traffic simulator SUMO [138]. Interaction with running simulations is achieved using Python scripts and the SUMO packages *TraCI* and *Sumolib*. The general setup used in our simulations is as follows.

- In all our experiments, we make use of the area in Dublin, Republic of Ireland shown in Figure B.3 (roughly speaking, south Dublin city centre to the canal), with all the U-turns removed from the network file, and a total of $S = 3,663$ states.
- A number of roads are selected as origins, destinations, and sources of congestion (see Figure B.3). In all experiments we use the set $\{O, D\}$ as an origin-destination (OD) pair. We use $C1$ in Experiment 1 and 2 and $\{C1, C2\}$ in Experiment 3 to simulate traffic jams on them.
- In all simulations we use a new vehicle type based on the default SUMO vehicle type[34] with maximum speed = 118.8 km/h and impatience = 0.5.
- For the generation of traffic jams, we modify the maximum speed of certain cars to be 6.12 km/h and populate the selected roads with them. When these vehicles are in possession of a token, they become virtual MUBEV vehicles.
- Whenever required, the shortest path is obtained via SUMO using the default routing algorithm (*dijkstra* [139]).
- We refer to a state of an agent as a set of road sections (see Figure B.4) and to a token trip as an RL episode.
- We set $H = 85$ for the length of the MDP's time horizon[35].

To reduce the size of the state space, we pre-process the road network to allow the merging of different road links into one state: for a given road link, whenever there is only one incoming direction and only one outgoing direction with neighbour road links, we join these edges into a single state (as shown in Figure B.4). With this preparatory step, the map shown in Figure B.3 that contains 10,803 road links results in a graph with only 3,663 states.

Concerning the design parameters of the reward funcion and the MUBEV algorithm, in all our experiments we set $\omega_D = \omega_T = 1$, $r_{max} = 1$, $\delta = 1$, and tuned the other design parameters as follows: $\alpha = 1.2$, $\beta = 1.3$, $\Omega = 20$. All the experiments rely on a number of background vehicles with random routes which will potentially carry tokens if required. Any specific additional setup for each individual experiment will be described in the corresponding subsection.

[34] `https://sumo.dlr.de/wiki/Definition_of_Vehicles_Vehicle_Types_and_Routes`

[35] This number corresponds to the length of the largest shortest path between an arbitrary origin and destination D (55 transitions in our case) plus a degree of freedom of 30 transitions to properly cope with uncertainties.

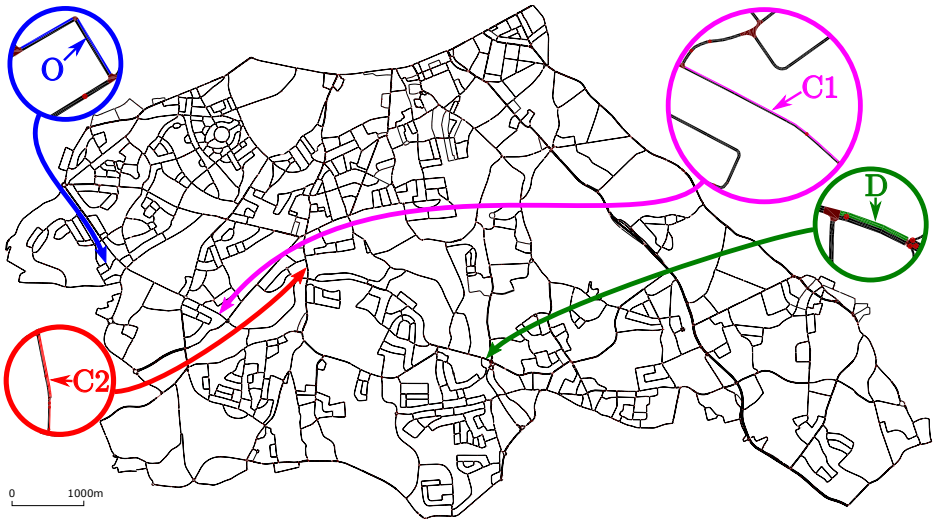


Figure B.3: Realistic road network used in the experiments: a part of Dublin, Republic of Ireland. Four road segments of interest are highlighted, namely O, D, $C1$ and $C2$.

B.4.1 Experiment 1: Optimal Route Estimation Under Uncertainty

The purpose of the first experiment is to determine if our DLT-enabled RL approach can estimate a simple unknown environment. To this end, we define the following experiment. We specify an OD pair for which shortest path is known and then, at various time instances, artificially introduce congestion along the shortest path. For this scenario, we should show that the token-enabled MUBEV algorithm can distinguish between these two situations, and, in the case of congestion, find the next best route between origin and destination.

Specifically, this first experiment is conducted as follows. We use a single token over each episode of the learning process, meaning that data from the token over every episode is used to update the MUBEV policy. For this, the MUBEV token has a fixed OD pair, and we select the road section labelled as $C1$ (which belongs to the shortest path for the selected OD pair) to generate a traffic jam on it at different intervals (see Figure B.3). Then, over each new episode we start the token from O and ask it to travel to D, keeping a record of its performance in terms of travel distance (route length) and travel time regardless of its success in attempting to reach D. Additionally, a token has a maximum number of allowed links (defined by the length of the MDP's time horizon) that it can traverse, and if it does not reach its destination within this restriction, then the token trip is declared incomplete (i.e., unsuccessful). The results for this experiment are shown in Figure B.5, from which we can draw two main conclusions:

- in general, we can see that the token succeeds in both avoiding the traffic jam once congestion is created, and returning to the shortest path once congestion is removed, using a reasonably small number of episodes (see Figure B.5 bottom);
- as time passes, more information is collected from the environment in the form of reward, and the token is more likely to fully complete a trip for the given OD pair (i.e., fewer red crosses are obtained as the experiment progresses in Figure B.5).

These two observations validate our expectations about the UBEV-based routing system: (i) it is able to adapt to uncertain environments, and (ii) its performance improves as time passes. It is worth noting that this experiment is useful to analyse the performance of a single token in the iterative learning process from the environment using a fixed OD pair.

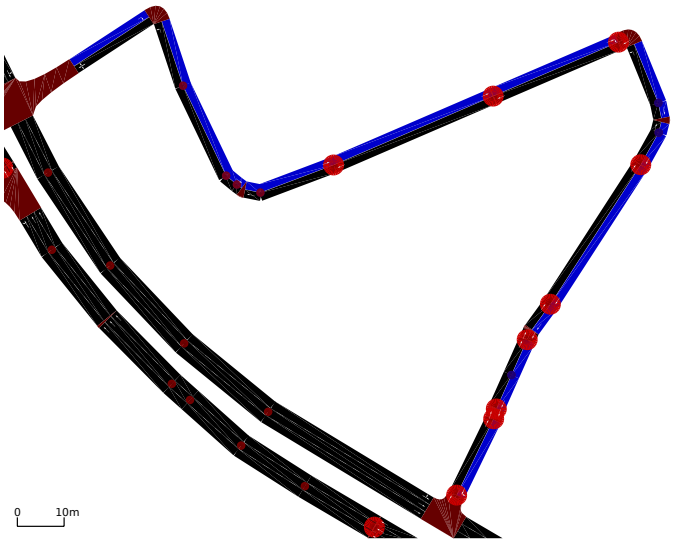


Figure B.4: State model: a state corresponds to a set of road links. Road links marked in blue are merged into one state.

B.4.2 Experiment 2: Optimal Route Planning Under Multiple Uncertainties.

Now we want to evaluate optimal routing under multiple uncertainties. For this, we use a similar setup as in Experiment 1 (i.e., same OD pair, intermittent traffic jam on $C1$, and one MUBEV token probing the environment), and additionally include a second traffic jam using the following procedure: 1) traffic jam on $C1$ is introduced, 2) the system learns the optimal detour, and 3) second traffic jam is introduced on such an optimal detour (specifically on road link $C2$, as seen in Figure B.3). The results of a single realisation of this procedure are shown in Figure B.6.

As can be seen from Figure B.6, a new optimal detour can be learnt after the second traffic jam is created, and once the two congestions are removed the system rapidly returns to the shortest path policy. Recall that red marks represent uncompleted routes (i.e., the destination state is not reached within an episode), which are more likely to appear when congestions are introduced.

Note that once the environment has been learned, the resulting route recommendations gleaned from the environment can be made available to the wider community of vehicles. We explore this in the following experiment.

B.4.3 Experiment 3: Route Recommendations from the UBEV-Based System and Speedup in Learning.

The previous experiments are a simple demonstration of the use of MUBEV in a mobility context. We now explore a scenario where multiple tokens, starting from different origins, are used to update the MDP's policy over each episode. Specifically, in this third experiment, we evaluate the performance of MUBEV as a function of the number of tokens over each episode, subject to a uniform geographical distribution of origins and a common destination (namely, road link D). Additionally, we analyse the performance of a *test* (non-MUBEV) *vehicle* trying to reach destination D from the given fixed origin O, using a recommendation from a simplistic UBEV-based routing system. In this case, the initial recommendation corresponds to the shortest path policy, and further recommendations come from the refinement of such a policy. In addition, if a complete route cannot be calculated using the MUBEV recommender system, then the most recent valid recommendation is reused. Remember that the MDP's policy is updated at the end of each episode, and hence we only release a new test vehicle at the end of each episode once the policy

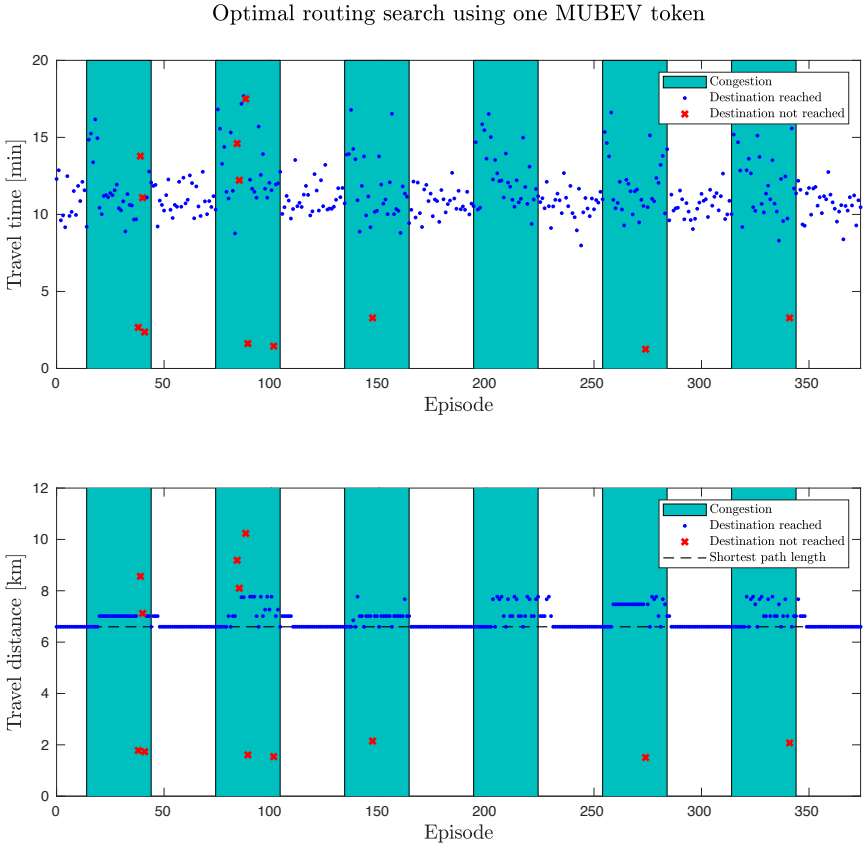


Figure B.5: Experiment 1: travel time and travel distance of a single token during the iterative learning process on a changing environment, using a fixed OD pair and approaching an intermittently congested road link. Each datapoint corresponds to information registered at the end of each episode (i.e., trip).

has been updated. The results for this experiment are depicted in Figure B.7 and Figure B.8.

In Figures B.7 and B.8, it can be observed that the number of participating tokens directly affects the convergence rate of the algorithm. As expected, the more tokens involved, the faster the learning process. Clearly, these values cannot reach zero as at least one episode of learning is required regardless of how many tokens are used to explore the environment.

B.5 Summary

We introduced a DLT design for smart mobility applications. The objectives of the DLT are: (i) preserving the privacy of the individuals, including General Data Protection Regulation (GDPR) compliance; (ii) enabling individuals to retain ownership of their own data; (iiii) enabling consumers and regulatory agencies alike to confirm the origin, veracity, and legal ownership of data, products and services; and (iv) securing such data sets from misuse by malevolent actors. As a use case of the proposed approach, we successfuly presented a DLT-supported distributed RL algorithm to determine an unknown distribution of traffic patterns in a city.

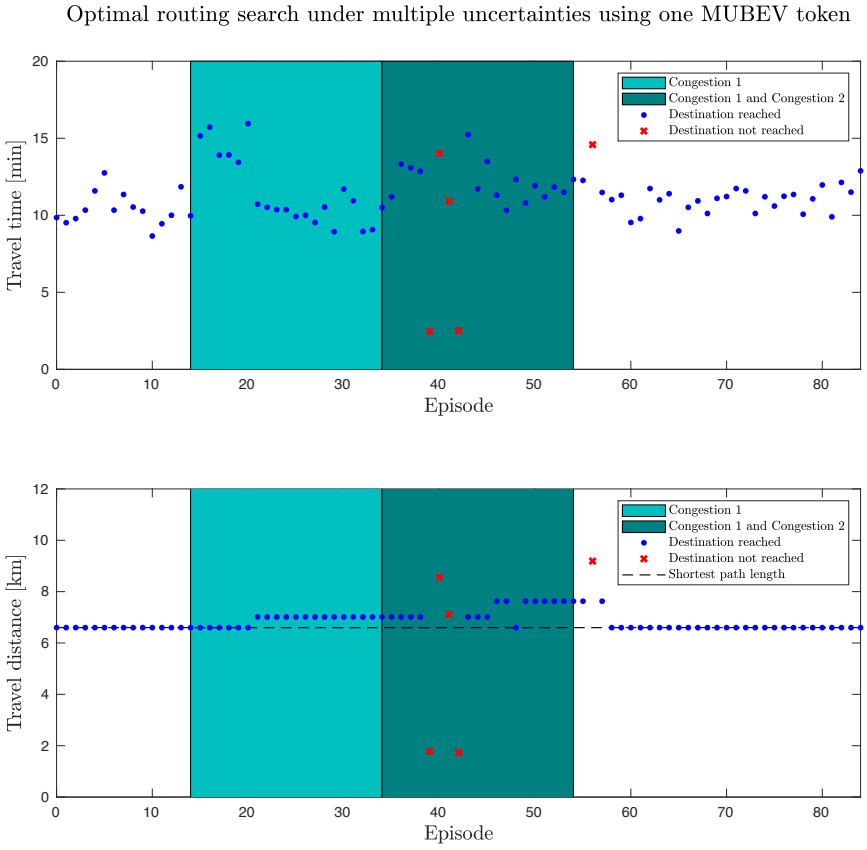


Figure B.6: Experiment 2: travel time and travel distance of a single token using a fixed OD pair during the iterative learning process with multiple uncertainties on the environment (two intermittently congested road links). Each datapoint corresponds to information registered at the end of each episode (i.e., trip).

Algorithm 14 Modified **U**pper **B**ounding the **E**xpected Next State **V**alue (**UBEV**) Algorithm - MUBEV

Require: $\mathcal{S}$; $\mathcal{A}$; H; $\mathbf{P}$; Π_{SP}; M; $\delta \in (0,1]$; r_{max}.

$n(s,a,t) = R(s,a,t) = 0$; $\hat{V}(s,t') = 0$; $\hat{Q}(s,a,t') = 0$
$\forall s, s' \in \mathcal{S},\ a \in \mathcal{A}_s, t \in \mathcal{H},\ t' \in \{1, 2, \ldots, H+1\}$.

$\delta' = \delta/9$; $V_{max} = H * r_{max}$.

for $k = 1, 2, 3...$ **do**
  // Optimistic planning loop
  for $t = H$ **to** 1 **do**
    $\hat{V}_{t+1} = \hat{V}(\cdot, t+1)$
    $\tilde{V}_{min} = \min\left(\min(\hat{V}_{t+1}), V_{max}\right)$
    $\tilde{V}_{max} = \min\left(\max(\hat{V}_{t+1}), V_{max}\right)$
    $\Delta\tilde{V} = \tilde{V}_{max} - \tilde{V}_{min}$
    for $s \in \mathcal{S}$ **do**
      for $a \in \mathcal{A}_s$ **do**
        $r = r_{max}$; $EV = \tilde{V}_{max}$
        if $n(s,a,t) > 0$ **then**
          $\eta_1 = 2 \ln \ln (\max(e, n(s,a,t)))$
          $\eta_2 = \ln(18 * S * A_s * H/\delta')$
          $\phi = \sqrt{\frac{\eta_1 + \eta_2}{n(s,a,t)}}$
          $\hat{V}_{next} = P(\cdot, s, a, t) \times \hat{V}_{t+1}$
          $EV = \min\left(\tilde{V}_{max}, \hat{V}_{next} + \phi * \Delta\tilde{V}\right)$
          $\hat{r}(s,a,t) = \frac{R(s,a,t)}{n(s,a,t)}$
          $r = \min(r_{max}, \hat{r} + \phi)$
        end if
        $Q(a) = r + EV$
      end for
      $\hat{Q}(s, \cdot, t) = Q$;
      if $Q_i = Q_j\ \forall Q_i, Q_j \in Q$ **then**
        $\tilde{a} = \Pi_{SP}(s,t)$
      else
        $\tilde{a} = \arg\max_{a \in \mathcal{A}_s} Q(a)$
      end if
      $\pi_k(s,t) = \tilde{a}$; $\hat{V}(s,t) = Q(\tilde{a})$
    end for
  end for
  // Execute policy for one episode
  $\tilde{s} = \left[s_1^{(1)}, ..., s_1^{(M)}\right] \sim \mathcal{U}(1, S),\ s_1^{(i)} \neq s_1^{(j)}\ \forall i, j \in [1, M]$
  for $t = 1$ **to** H **do**
    $a_t^{(m)} = \pi_k(s_t^{(m)}, t)$
    $s_{t+1}^{(m)} \sim P(\cdot | s_t^{(m)}, a_t^{(m)}, t)$
    $r_t = \mathcal{R}\left(s_t^{(m)}, s_{t+1}^{(m)}\right)$ ▷ Call to Function 1
    $R\left(s_t^{(m)}, a_t^{(m)}, t'\right) += r_t,\ \forall t' \in \mathcal{H}$
    $n\left(s_t^{(m)}, a_t^{(m)}, t'\right) ++,\ \forall t' \in \mathcal{H}$
  end for
end for

Function 1 The Reward Function

Require: s_t; s_{t+1}; $\tau(s_{t+1})$; α; β; w_D; w_T; Ω; r_{max}.
Ensure: r_t.

function $\mathcal{R}(s_t; s_{t+1})$
  if $s_{t+1} \neq s_t$ **then**
    // Distance reward computation
    $d = D(s_t) - L(s_t)$
    if $d \neq 0$ **then**
      $r_D = r_{max} - \frac{D(s_{t+1})}{d}$
    else
      $r_D = r_{max}$
    end if
    // Time reward computation
    $\tau_{ref} = RY(s_{t+1}) + \alpha * \tau_{min}(s_{t+1})$
    if $\tau(s_{t+1}) \leq \tau_{ref}$ **or** $s_{t+1} = s_f$ **then**
      $r_T = 0$
    else
      $r_T = -\beta * \frac{\tau(s_{t+1})}{\tau_{ref}}$
      // Applying the edge coefficient
      if $RY(s_{t+1}) = 0$ **then**
        $r_T = r_T * EC(s_{t+1})$
      end if
    end if
    $r_t = w_D * r_D + w_T * r_T$ ▷ Total reward
  else ▷ Jumping to the same state
    if $s_{t+1} \neq s_f$ **then**
      // Penalty: staying at not destination
      $r_t = -\Omega$
    else
      $r_t = r_{max}$
    end if
  end ifreturn r_t
end function

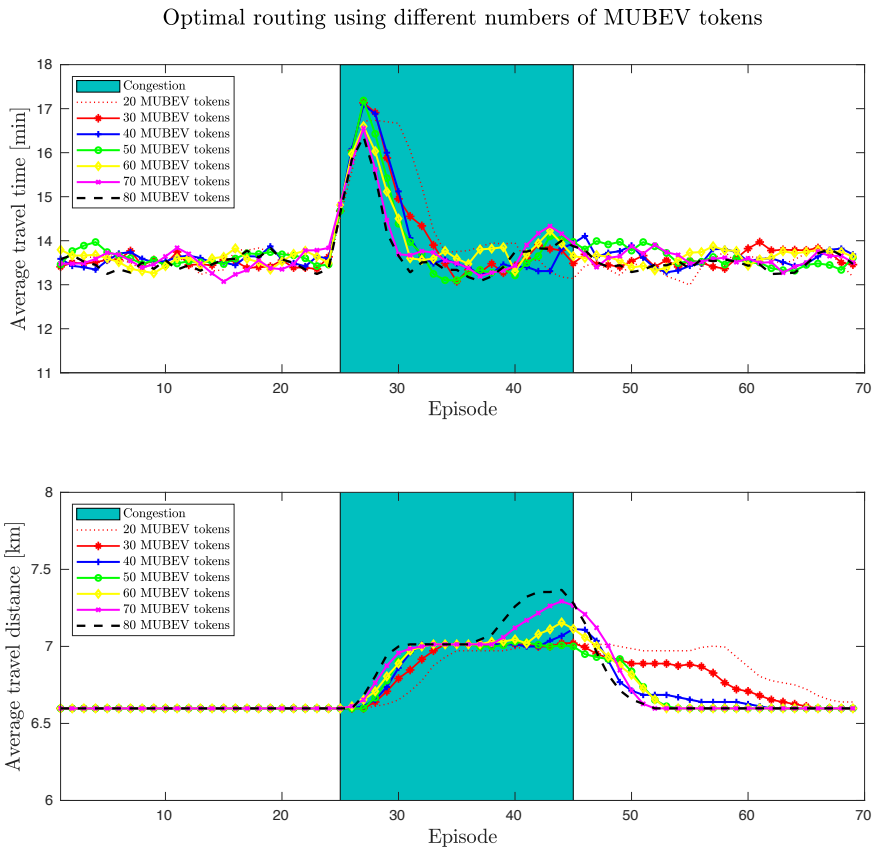


Figure B.7: Experiment 3: average travel time/distance of a test vehicle using route recommendations from a UBEV-based routing system involving multiple MUBEV tokens. Each datapoint corresponds to the average value collected at the end of each episode from 10 different realisations of the experiment, and a moving average with window size 2 was later used to smooth the resulting signals.

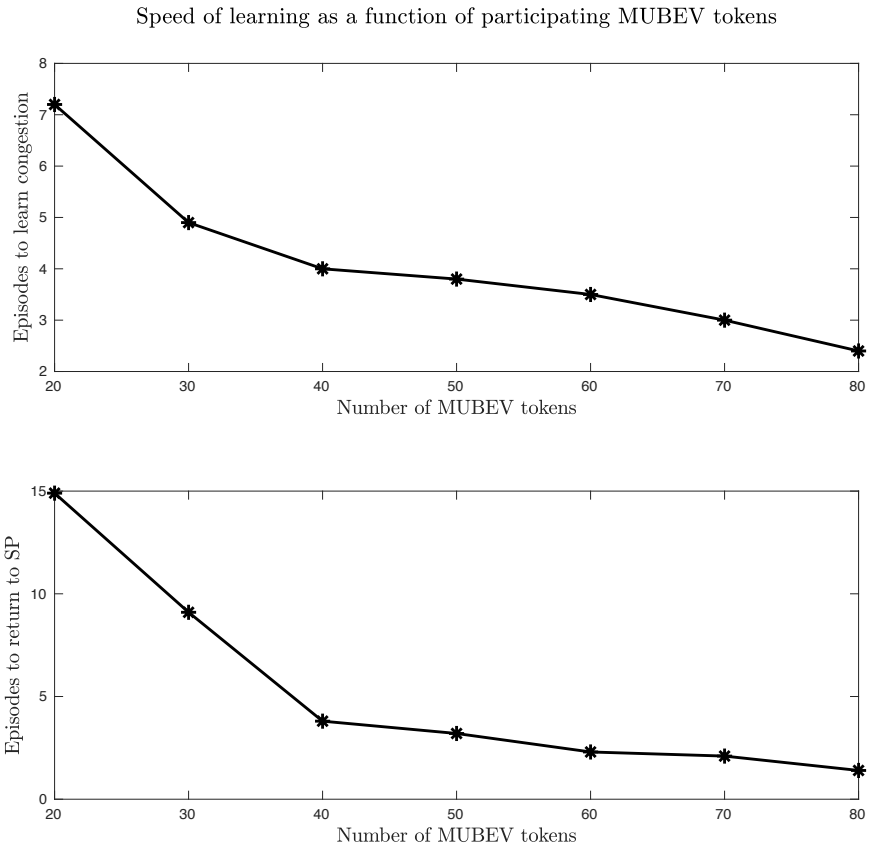


Figure B.8: Experiment 3: average learning speed using multiple MUBEV tokens. Each datapoint corresponds to the average value obtained at the end of 10 different realisations of the experiment.

Appendix C

Distributed Random Number Generation

***Abstract*—The work presented in this appendix is an excerpt from a report produced for IOTA Foundation as part of a Summer internship. It outlines the importance of trustworthy shared randomness in the IOTA protocol and reviews the state of the art for distributed random number generation. The report concludes with a proposal for a random number generator to be used in the IOTA DLT network.**

C.1 Introduction

Reliable randomness is essential to many distributed consensus protocols, and to the operation of many distributed systems. Examples include early randomised consensus protocols employing a common coin to achieve consensus in the Byzantine setting, and more recent PoS systems which often require agreement on a random number to be used for leader election. A random element in a distributed system has a clear motivation—attackers can't influence the progression of the protocol as easily if they can't predict what is going to happen next, and randomness can play a part in many aspects of a distributed system, as we shall discuss here. But the question of whether an attacker can predict what will happen next depends on the source of the randomness. How do nodes agree on a random number? Do they all receive a random number from some trusted central entity? Do they use some distributed protocol in which each node contributes to the randomness? Or perhaps something in between, in which a number of high reputation nodes or a consortium of some kind produces a random number to be used by all other nodes. Each of these approaches has its advantages and drawbacks.

Following the removal of the Coordinator from the IOTA Network (a node operated by IOTA Foundation which issues milestones and controls all confirmation), nodes will need to decide on confirmation of transactions without the stamp of approval from the IOTA Foundation's node. The solution to this problem is outlined in the *Coordicide* white paper [44], and contains a number of ingredients—layers of finality on top of the existing tip selection algorithms and fundamental DAG-structure of the ledger. One of the proposed protocols for deciding on conflicts is known as Fast Probabilistic Consensus (FPC) [46, 74] which has been shown to achieve consensus in the Byzantine setting with high probability. FPC requires a random number to be agreed upon as a threshold for deciding upon opinions on transactions in each round of FPC voting. This threshold must not be able to be predicted by adversaries, or it could be possible to delay consensus and could make the DAG less secure against forks. Although FPC is the main reason the IOTA network needs shared random numbers at the time of writing, a trusted random beacon could be very useful for a variety of other services for IOTA in the future. There are a number of approaches to producing

random numbers, with varying degrees of decentralisation.

One option (although perhaps not a very good one) is to use a centralised beacon to obtain random numbers. The advantage of this approach is that it requires practically no additional work for nodes—all they would have to do is query the beacon for a fresh random number when it is required. IOTA Foundation could operate a reliable beacon which satisfies all of the above properties, and indeed this could be done in a publicly verifiable manner. However, this solution is rather unappealing as there remains a degree of centralisation which we hope to elimenate by removing the coordinator. It would also be possible to use a beacon operated by another trusted organisation such as the NIST beacon[36], but concerns remain about the trustworthiness of such sources.

Ideally, we would like to distribute trust in the generation of this random number across participating nodes somehow. In this appendix, we present a number of methods for achieving this.

C.2 State of the Art

A distributed random number generator (dRNG) with the desired properties of being *available*, *unpredictable*, *unbiasable* and with light requirements from nodes is not a straightforward task. To motivate the need for careful thought, we begin by presenting a naive solution.

Perhaps the simplest approach one could take would be for everyone to generate a random number and send it to everyone else. When each node has a random value for every other node, they combine the values with a bitwise *exclusive or* operator. Provided everyone contributes a number at the same time without the knowledge of what others are committing, this protocol succeeds. However, in a realistic network, an attacker could simply wait until they have received values from all other nodes, and then select their value too such that the result will be precisely what they want.

Suppose we try to solve this problem with a commit-reveal approach, i.e., each node submits a hash of their random value (commitment), and after a suitable number of participants have committed, each participant reveals their value. However, the issue here is that a malicious participant could choose not to reveal their value, resulting in the protocol having to restart. The attacker could repeat this process until the protocol produces a number to their liking. Suppose instead that we require only a subset of the committed values to be revealed and used—a malicious actor could wait until only one more reveal is required before deciding whether they would reveal or not, allowing them to bias the random number to some extent.

The issues discussed here are a result of the “last actor" problem, and we need to employ some new cryptographic primitives to get around it. The solutions in the literature are mainly based around publicly verifiable secret sharing (PVSS) and threshold signature schemes such as TSS and TBLS. The basic algorithms here are very high in communication complexity, because they are essentially BFT agreement protocols. The optimisations and reductions in complexity are generally case-specific, e.g., we may need sequences of random numbers, or perhaps just a one-off number for a lottery draw. Other solutions have been based on proofs of delay, or verifiable delay functions (VDFs).

C.2.1 Threshold-Based Approaches

Threshold-based approaches are essentially a more advanced commit-and-reveal, where it does not matter which subset of the committers contribute to revealing. The random output commitment is generated in a distributed manner in the form of a secret which is shared among the n participants. For the random output to be revealed, a threshold t of the n participants must contribute their signature, so we can use a (t,n)-threshold signature scheme with $t=f+1$, where f is the maximum number of malicious actors that can be tolerated by the protocol. The issue with these approaches

[36] https://beacon.nist.gov/home

is that the allocation of shares requires either a trusted dealer or Byzantine agreement in the form of a distributed key generation (DKG) process. Some examples of these approaches are as follows.

Randshare, Randhound and Randherd[37] are described in [140]. Randshare is the name given to the distributed key generation protocol described. Randhound is a protocol for a client to request a random number from a group of servers. The Randhound client arranges the servers into disjoint groups and requests a random number from each group, which are then combined. This improves the scalability over doing one DKG among all servers. Randherd is a protocol aimed at producing a beacon in which a "cothority" divide in to groups randomly (seeded by a run of Randhound) and share secrets in the group which are aggregated by the Randherd leader. The protocol is slow and does not scale particularly well despite optimisations.

Drand is a more recent project of the authors of [140]. The cryptographic background of Drand is described concisely in a blog post[38] by a member of DEDIS research group. Drand is still in development, with improvements being made to the signature scheme and the curves used, and the code is available on the EPFL DEDIS group's github[39]. The protocol involves a DKG to generate public and private keys for each participant, and public and private shared keys. Once this process is complete, random numbers can be produced by signing a message which is the hash of the last random number produced—the first random number needs to be provided as a seed for this chain of random numbers, and can, for example, be produced by running another DKG for a once off random number, or a protocol similar to Randhound [140].

The League of Entropy Beacon[40] is a cothority random beacon project based on Drand. The cothority is made up of a number of trusted institutions from industry and academia, each of whom use their own local source of entropy for a Pederson DKG [141] which seeds the chained randomness of Drand. An example of one such source of entropy is *LavaRand*[41] from cloudflare, one of the cothority participants. LavaRand generates random numbers with high entropy using images captured continuously from a wall covered in Lava lamps in cloudflare HQ office.

DFINITY [6] provides a random beacon (referred to as *threshold relay chain*[42] [142]) as the backbone of the consensus mechanism for their "internet computer". Their beacon is also based on threshold signatures and relies on similar primitives to Drand. DFINITY is permissioned, and for random number generation, the nodes are divided into groups. Keys for the BLS signature scheme are generated with a Joint-Feldman DKG [141] in each group separately. The Joint-Feldman DKG is a simpler form of DKG that can be performed in a single round when all nodes participate, but it is known to be biasable, albeit in a way that does not generally weaken the hardness of the discrete-log problem. Note also that the grouping of signers allows the protocol as a whole to scale. As long as a t-majority of each group are honest, a group can issue a signature and produce a random number, similarly to Drand. However, the random numbers from each round are used to select the group who are to issue the random number in subsequent rounds. This grouping approach is a trade-off between scalability and security, because random numbers can be produced faster, but we rely on assumptions that each and every group has t honest participants. The increase in speed arises from the fact that for each random number, partial signatures only need to be gathered from t nodes, where t is less than the group size. However, the fact that the group for subsequent round is chosen at random based on random beacon values means that it is difficult to target attacks at specific groups effectively. The signature scheme used is based on Schnorr signatures in paper, but they have now upgraded implementation to BLS[43].

Hydrand [143] is not based on threshold cryptography, but is rather a Byzantine agreement protocol which has reduced communication complexity from $O(n^3)$ to $O(n^2)$. The protocol is a *propose*, *acknowledge*, *vote* structure in which each round has a leader. The leader is exempt

[37] https://github.com/dedis/cothority
[38] https://hackmd.io/@nikkolasg/HyUAgm234
[39] https://github.com/dedis/drand
[40] https://blog.cloudflare.com/league-of-entropy/
[41] https://blog.cloudflare.com/lavarand-in-production-the-nitty-gritty-technical-details/
[42] Go implementation available at https://github.com/dfinity/random-beacon
[43] https://github.com/dedis/cothority/tree/master/blscosi

from being leader for the next f rounds which provides a beacon in which unpredictability is probabilistically guaranteed, and guaranteed after f+1 rounds. The main relevance of this paper to this report is that it provides a good literature review with a comprehensive comparison of key features of some other distributed randomness beacons.

Algorand [4] is a cryptocurrency whose PoS-based consensus mechanism is implemented using leader elections which they refer to as cryptographic sortition. Algorand does not explicitly output a random beacon, but threshold cryptography primitives are employed to privately elect block proposers. The approach creates a chain of verifiable random numbers, just like in Drand and DFINITY i.e. the seed for each round is derived from the random output of the last round. In fact, it is not specified how the initial seed is to be generated in [4], but it is stated that some random number generation protocol will be required to be carried out by the initial users.

C.2.2 Delay-Based Approaches

Another solution to the last actor problem in the generation of randomness is to enforce a time delay between the commitments of contributions and the calculation of the resulting value. In other words, rather than XOR the contributions together, we concatenate all contributions and apply a VDF. Alternatively, a delay can be applied to a high entropy random source, such as block hashes in a blockchain or stock market prices. The below examples illustrate these delay-based approaches.

[144] presents "the random zoo" which refers to a collection of three modules which are proposed for generating random elliptic curve parameters. The *sloth* module is a VDF based on iteratively calculating modular square roots [73]. The *unicorn* module is a protocol which involves collecting contributions from the public and concatenating them all with a salt from a central authority. This salt could be issued by an IOTA Foundation node or a high-reputation node and could perhaps include information about the DAG at that time or some other high entropy source of randomness which will be combined with all the other public contributions. The *trx* module simply involves the application of unicorn to generation of elliptic curve parameters so is not directly relevant to this report.

[145] presents a protocol for applying VDFs to data from Ethereum [24] blocks to generate unpredictable randomness. VDFs are applied to block hashes to prevent even the most powerful miners from tampering with the random beacon value, because by the time the VDF output can be known, the block corresponding to the input is finalised.

C.3 IOTA dRNG

In this section, we provide recommendations for how a random number generator should be implemented as part of the IOTA protocol, referring to the techniques discussed in the above sections. The proposed approach is based heavily on Drand and DFINITY—we propose using threshold signatures and a feedback loop to create chained random numbers, and we propose a number of options for seeding the chain. Our reasoning for choosing this approach is that the other approaches based on threshold signatures, or those based on VDFs are all more computationally intense and require more time. However, we can still use some of these slowly, more computationally intense techniques to seed the chain of random numbers. We assume here that not all nodes will participate in random number generation, but rather a subset of nodes, which we shall refer to as dRNG nodes. We discuss how we might choose these nodes further below. Figure C.1 illustrates the high-level operation of the proposed dRNG.

C.3.1 Seeding the dRNG

The chain of random numbers we propose to create must be seeded with an initial random number, denoted as σ_0 in Figure C.1. The goal of the process of creating an initial condition with high entropy and this can be done in a variety of ways.

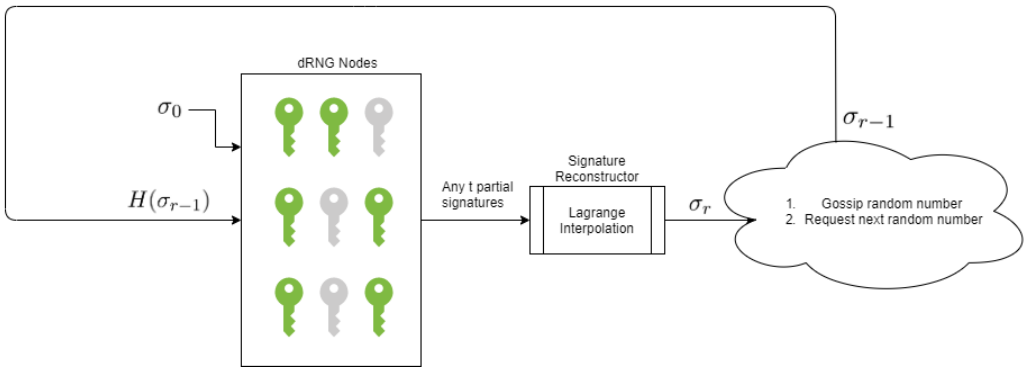


Figure C.1: Chained randomness generation with t-of-n threshold signatures.

Threshold DKG

We can use a t-of-n threshold DKG, and use the shared secret key as σ_0. This is similar to Randhound from [140]. The entropy of σ_0 depends on the entropy contributed by each of the participants. In the League of Entropy beacon, each of the participants has a local RNG which they can use for this DKG.

Entropy from the Community Using VDFs

VDFs can be used in a process similar to [144] to gather contributions of random numbers from the IOTA community on some public forum. The contributions would all be concatenated, along with a high entropy salt provided by IOTA Foundation (this does not need to be publicly verifiable, as it only adds entropy, but can't be used to influence the output), and a VDF applied to them to produce the first random number, σ_0. Every community member could easily verify that their contribution has been included.

Entropy from the DAG Using VDFs

Entropy from the DAG can be derived from a number of transactions that have been fully agreed upon (e.g., milestones), and the protocol from [145] employed to apply a VDF and produce a random σ_0.

C.3.2 Chained Randomness

Once a high entropy seed, σ_0, has been produced, we propose using threshold signatures to produce a chain of random numbers in a fast, non-interactive manner.

Distributed Key Generation

This phase only needs to be complete once for a group of n dRNG nodes. The dRNG nodes will be discussed further below, but they may not need to change very often. The signature scheme we propose to use is BLS [146, 147], as signatures are significantly shorter than the popular alternative, Schnorr signatures [148], with the slight trade-off that the pairing operation required for verification of signatures can be computationally intensive.

Partial Signature Generation

To generate a partial signature which will be combined with other partial signatures to compute the next random number, a dRNG node computes a hash of the last random number, and computes a signature using their partial secret key, s_i.

$$\sigma_r(i) = s_i H(r||\sigma_{r-1}) \tag{C.1}$$

where $\sigma_r(i)$ denotes the partial signature of dRNG node i for the r^{th} random number in the chain, and σ_{r-1} denotes the reconstructed full signature from round $r-1$. Note that the full signature, σ_r, serves as the random beacon value for round r.

Signature Reconstruction and Distribution

In order to reconstruct a full signature, which will be the next random beacon value, any t of the partial signatures, $\sigma_r(i)$, must be gathered from the dRNG nodes, and a Lagrange interpolation performed to reconstruct the full threshold signature, σ_r. This interpolation can be carried out by any node(s) and the reconstructed signature distributed as needed.

Signature Verification

Verification of the threshold signature, σ_r, requires the verifier to compute two pairings and verify that

$$e\left(H\left(r||\sigma_{r-1}\right), S\right) = e\left(\sigma_r, g_2\right) \tag{C.2}$$

where $e(\cdot)$ is a pairing function, S is the collective public key generated in the DKG, and g_2 is the generator of a bilinear group, $\mathbb{G}_2$, to which the shared public key belongs. Note that a similar verification step can be carried out on partial signatures prior to the reconstruction of the threshold signature.

C.3.3 Choosing dRNG Nodes

The dRNG nodes (the n nodes participating in the random number generation) should be carefully chosen, as any t colluding adversarial nodes chosen to be dRNG nodes could influence the random numbers produced without being detected.

n Highest Repuation Nodes

The n highest reputation nodes seem to be a good candidate as trustworthy dRNG nodes. It can also be assumed that nodes with high reputation are well connected and can hence propagate new random numbers through the network faster. n and t must be chosen and the n highest reputation nodes would change over time, so when some limit of k nodes are no longer in the top n, we would need to invoke a new DKG to update the dRNG nodes. As mentioned above, DKG takes a significant amount of time and could not be run regularly.

Industry Partners as dRNG Nodes

A more static and centralised solution would be to designate a number of trusted dRNG nodes to be run by industry partners of IOTA Foundation such as Volkswagen, Jaguar Land Rover and Bosch. This would produce a somewhat centralised random beacon similar in spirit to that of *The League of Entropy*. It is likely that such organisations will want to run nodes in the future and could reasonably take on the additional task of generating random numbers. This solution, although somewhat more centralised, has the advantage that DKG would only need to be run in the event of new partners joining or old partners dropping out.

Appendix D

Milestone-based results for Chapter 5

Here we present equivalent results for a number of plots presented in Chapter 5, but here we use milestone-based confirmation (Algorithm 8) rather than weight-based confirmation (Algorithm 7). These results are not as effective for visually illustrating satisfaction of the algorithm requirements as their weight-based counterparts due to the periodic nature of milestones, but they are of interest all the same.

Table D.1 gives the access control parameters for the first set of milestone-based simulations which correspond to the honest environment simulations of Section 5.3.1. In this milestone-based setting, node 0 (the highest reputation best-effort node illustrated in Figure 5.3) issues a milestone block every 10 seconds, and everything in this milestone's past cone can be marked as confirmed.

Table D.1: Access control algorithm parameters with milestone-based confirmation.

Scheduler			**Rate Setter**				**Buffer Man.**	**Tip Set Man.**	**Conf. Man.**
ν	Q_i	$DC_{\max}$	A	β	τ	W	$W_{\max}$	$h_{\max}$	T_{MS}
250	$\frac{rep_i}{\sum rep}$	1	0.075	0.7	0.2	1	500	3	10

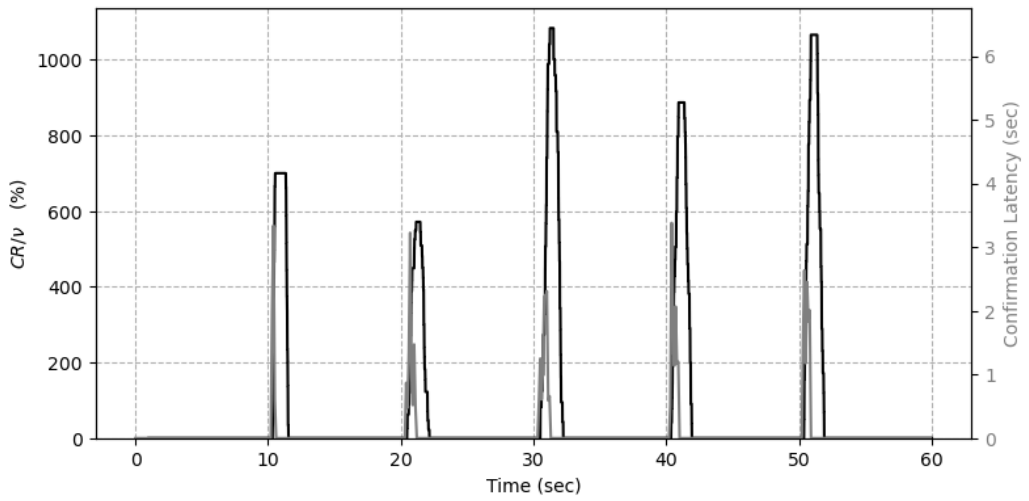


Figure D.1: Milestone-based confirmation: confirmation rate and mean confirmation latency over all blocks. Milestone-based equivalent of Figure 5.4.

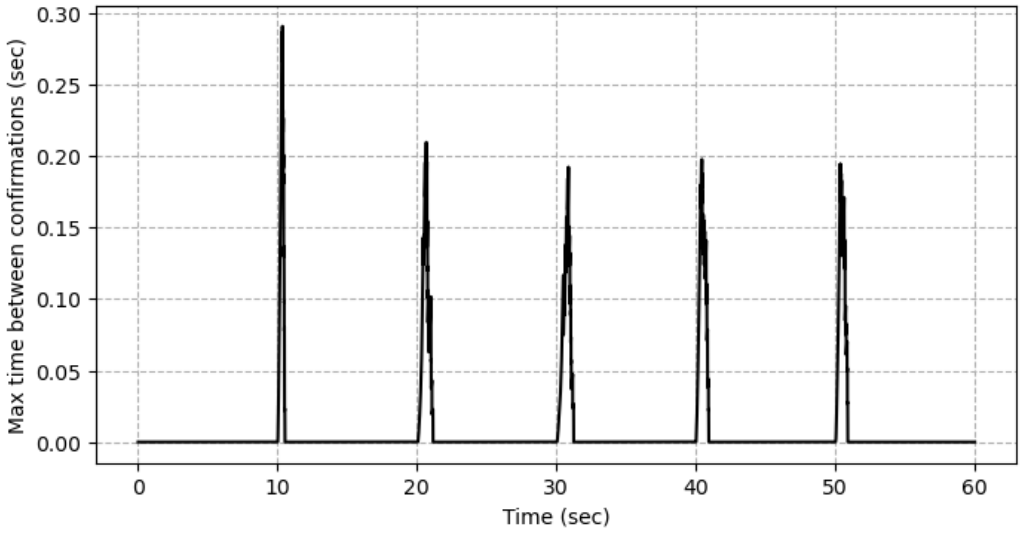


Figure D.2: Milestone-based confirmation: maximum time since partial confirmation for all partially confirmed blocks. Milestone-based equivalent of Figure 5.5.

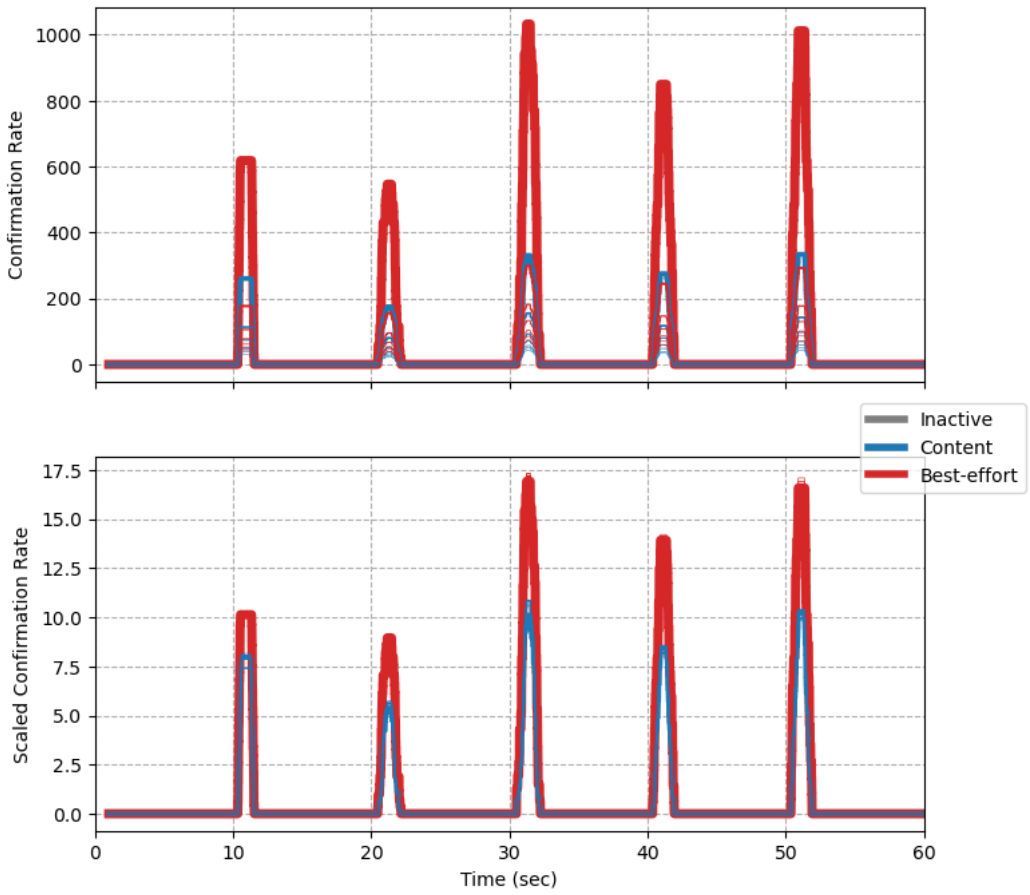


Figure D.3: Milestone-based confirmation: confirmation rate and scaled confirmation rate for each node. The bottom plot of scaled confirmation rate demonstrates that fairness in confirmation rate is achieved. Milestone-based equivalent of Figure 5.6.

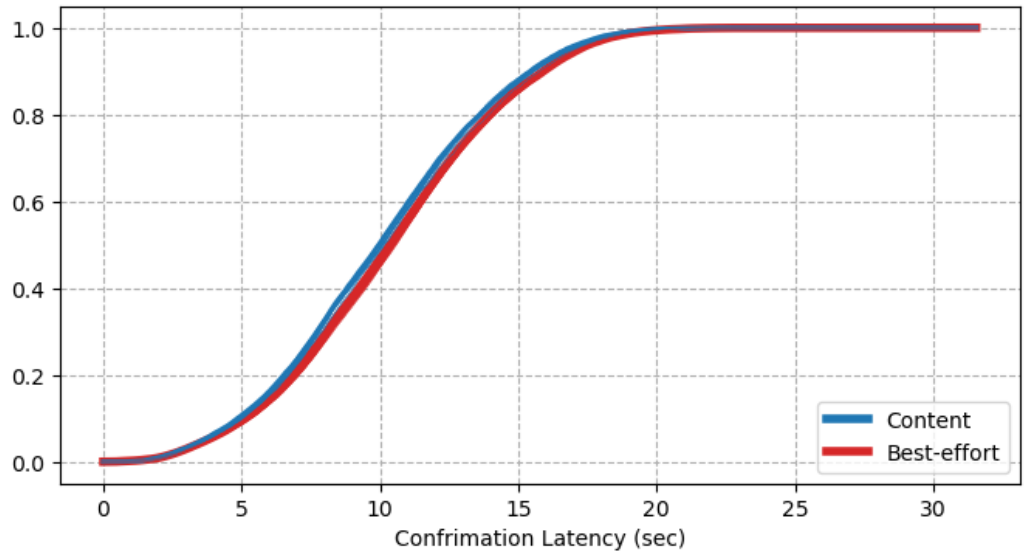


Figure D.4: Milestone-based confirmation: cumulative distribution of confirmation latency for each node. This demonstrates that fairness in confirmation latency is achieved. Milestone-based equivalent of Figure 5.7.

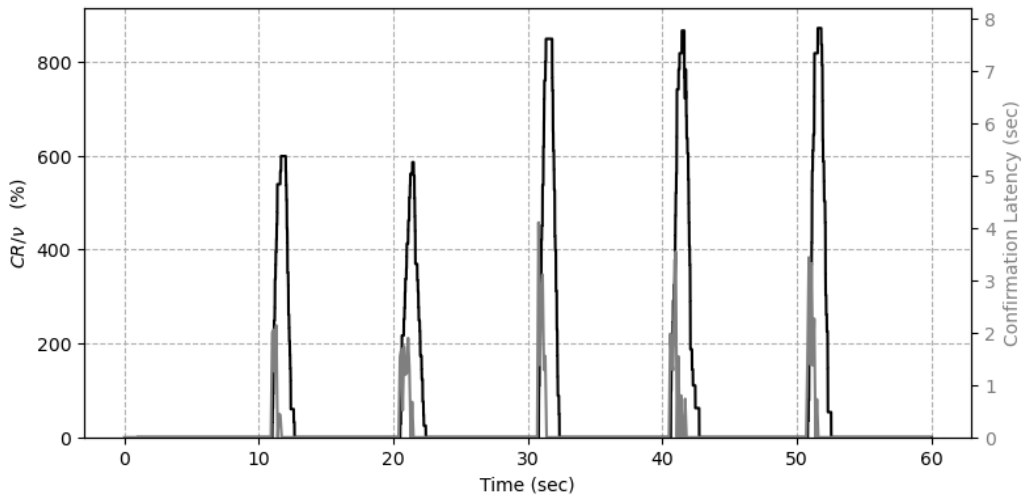


Figure D.5: Milestone-based confirmation: confirmation rate and mean confirmation latency over all blocks. Milestone-based equivalent of Figure 5.13.

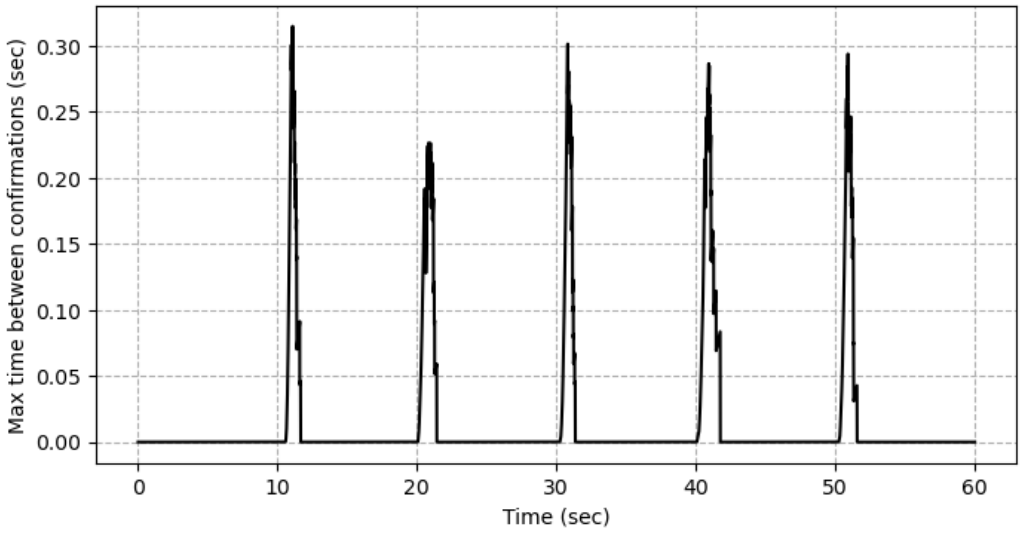


Figure D.6: Milestone-based confirmation: maximum time since partial confirmation for all partially confirmed blocks. Milestone-based equivalent of Figure 5.11.

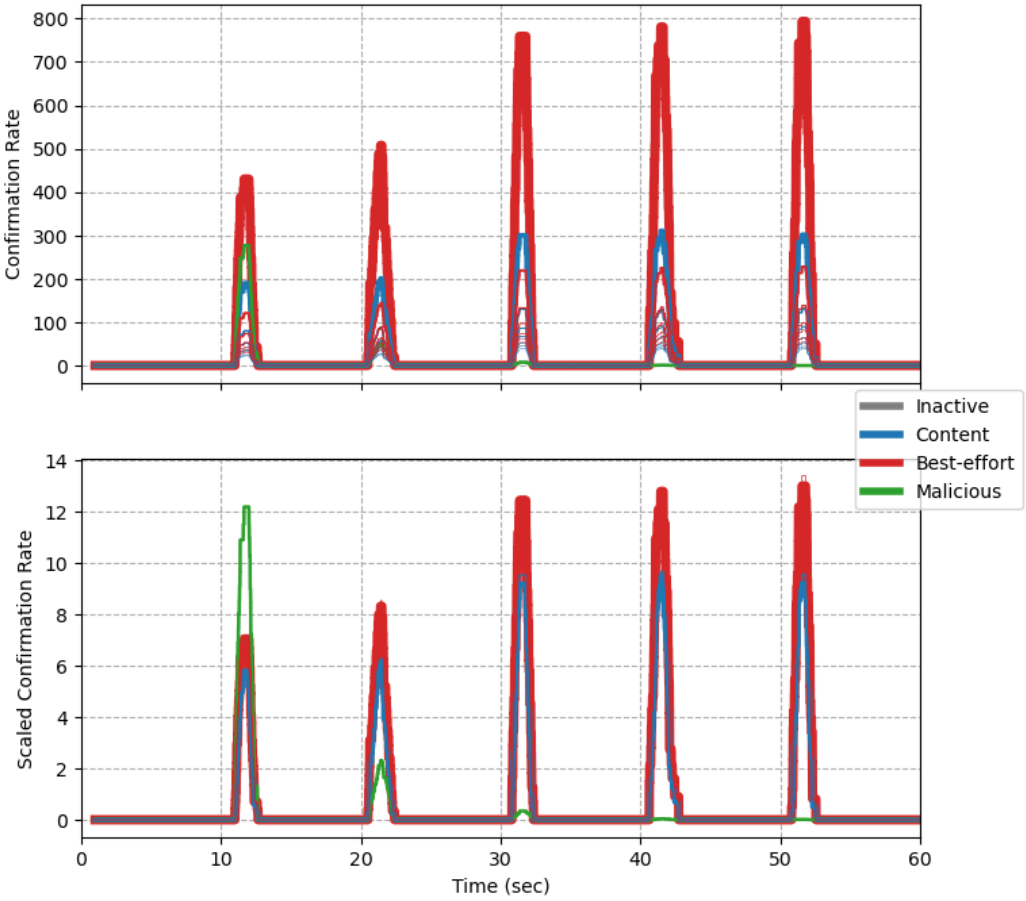


Figure D.7: Milestone-based confirmation: confirmation rate and scaled confirmation rate for each node. The bottom plot of scaled confirmation rate demonstrates that fairness in confirmation rate is achieved. Milestone-based equivalent of Figure 5.12.

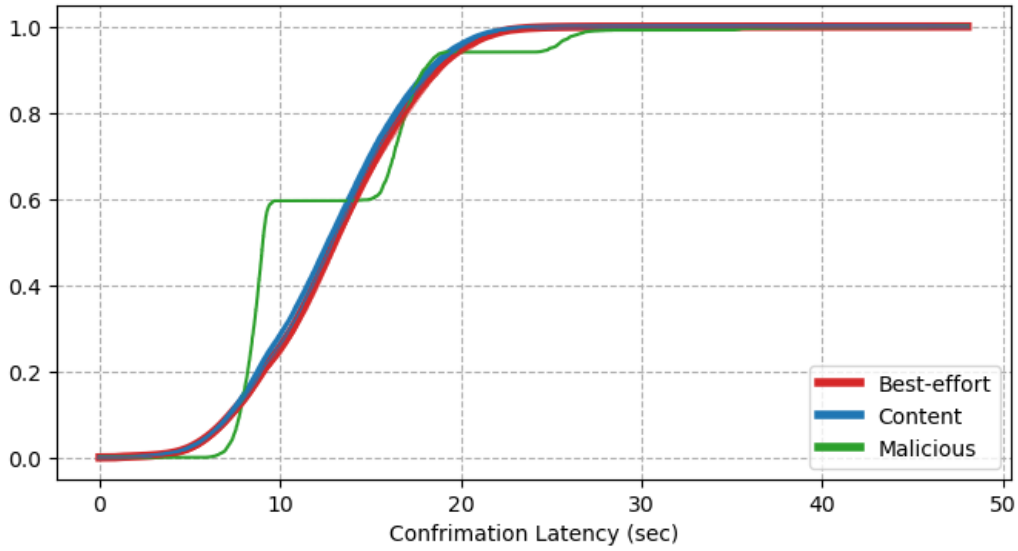


Figure D.8: Milestone-based confirmation: cumulative distribution of confirmation latency for each node. This demonstrates that fairness in confirmation latency is achieved. Milestone-based equivalent of Figure 5.13.