Master's Thesis

Healthor: Heterogeneity-aware Flow Control in DLTs to Increase Performance and Decentralization

Author: Jonas Theis (2625168)

1st supervisor: dr. ir. Animesh Trivedi
2nd supervisor: dr. Lin Wang
daily supervisor: dr. Luigi Vigneri (IOTA Foundation)

A thesis submitted in fulfillment of the requirements for the Master of Science degree in Parallel and Distributed Computer Systems

March 19, 2021

"The blockchain world is really nothing but an application of classical distributed computing where they've changed all the names to make it look different. [...] In classical distributed computing our adversary is kind of cute and sort of harmless. [...] In the modern world our adversary is much more frightening, an evil hacker."

— Maurice Herlihy (1)

Abstract

Permissionless distributed ledger technologies (DLTs) are gaining more and more popularity, especially for trustless money transactions and smart contracts, but have shortcomings in terms of performance, scalability, and energy efficiency. Reputation-based DLTs have been proposed to overcome these limitations of traditional DLTs and to enable feeless messages to power the machine-to-machine economy. These DLTs allow machines with widely heterogeneous capabilities to actively participate in message generation and consensus. However, the open nature of such DLTs can lead to centralization of decision-making power, e.g., by implicitly excluding slow participants through fast operation, thus defeating the purpose of building a decentralized network.

In this thesis, we introduce Healthor, a novel heterogeneity-aware flow-control mechanism for permissionless reputation-based DLTs. Healthor formalizes node heterogeneity by defining a health value as a function of its incoming message queue occupancy. We show that health signals can be used effectively by neighboring nodes to dynamically flow control messages while maintaining high decentralization. We perform extensive simulations, and show a 23% increase in throughput, a 76% decrease in latency, and four times increased node participation in consensus compared to state of the art. To the best of our knowledge, Healthor is the first system to systematically explore the ramifications of heterogeneity on DLTs and proposes a dynamic, heterogeneity-aware flow control. Healthor's source code (`https://github.com/jonastheis/healthor`) and simulation result data set (`https://zenodo.org/record/4573698`) are both publicly available.

Acknowledgements

In late 2016 I got interested in DLTs, started researching early projects, and looked closer into Bitcoin's mechanisms. I was and still am fascinated by the many facets and possibilities of this new technology: interesting and challenging problems and the potential to disrupt our trust-dependent, centralized world in terms of data ownership and economics. At the beginning of 2017, I came to know about IOTA via my friend Tilman who recommended *looking into it*. I did, was intrigued by the ideas, and kept following along with the progress. At the time I decided to learn the fundamentals of distributed computing as I understood DLTs to be an application of this established field. Eventually, the research master's degree *Parallel and Distributed Computer Systems* at VU Amsterdam caught my attention and seemed perfect for me. Fast-forward a few years: here I am completing the degree in cooperation with the IOTA Foundation.

I would like to thank the IOTA Foundation for the opportunity to become part of such an incredible team and learn from all my colleagues. Especially to Luigi Vigneri, my supervisor from the IOTA Foundation: Thank you for your continuous support, feedback, and valuable input and ideas.

To my supervisors from VU Amsterdam Animesh Trivedi and Lin Wang: Thank you so much for teaching me in your courses, our weekly meetings, and the constructive feedback. I have learned a lot, especially about scientific writing in the context of the thesis and our papers.

Much of this thesis work has been done while living together with my good friend Kian on the Canary Islands. Thanks for your willingness to listen, the discussions, and the encouragement I needed from time to time.

Contents

List of Figures

LIST OF FIGURES

LIST OF FIGURES

List of Tables

Glossary

DAG	Directed acyclic graph
DLT	Distributed ledger technology
MPS	Messages per second
P2P	Peer-to-peer
PoS	Proof of stake
PoW	Proof of work
VM	Virtual machine

Chapter 1

Introduction

With its inception in 2008, Bitcoin has sparked a whole new world of distributed ledger technologies (DLTs) (4, 5). DLTs are gaining popularity ever since, especially for trustless money transactions and code execution[1] enabling the recent trend towards decentralized finance with stablecoins, decentralized exchanges, and decentralized lending (6).

Conceptually, a distributed ledger is an immutable, replicated, and shared data structure that keeps track of ledger state entries, e.g., monetary transactions, in a distributed system without the need for a centralized authority but instead utilizes a distributed consensus mechanism (5, 7). Ledger state updates are disseminated using a peer-to-peer (P2P) network between ledger participating nodes (8, 9, 10). Theoretically, a permissionless DLT is open for anyone to join and keep track of the ledger and participate in consensus (9).

However, in practice traditional DLTs present limitations in terms of not only decentralization but also scalability, performance, and energy efficiency (11, 12, 13). Hence, a number of alternatives have been proposed in the last few years (14, 15, 16).

1.1 Permissionless Reputation-based DLTs Enable Heterogeneity

In this thesis, we focus on *permissionless reputation-based* DLTs (10, 17, 18, 19, 20, 21), which achieve consensus through voting instead of expensive mining races. Reputation-based DLTs *can* enable a wide range of new application domains, such as the machine-to-machine economy for the Internet of Things or public transparent supply chains (10, 20, 21), by overcoming Bitcoin's limitations: network throughput is not constrained at the protocol level, feeless messages are possible, low-power devices can participate in the consensus, etc.

[1] Trustless code execution is usually referred to as *smart contracts* and enables decentralized applications to be built.

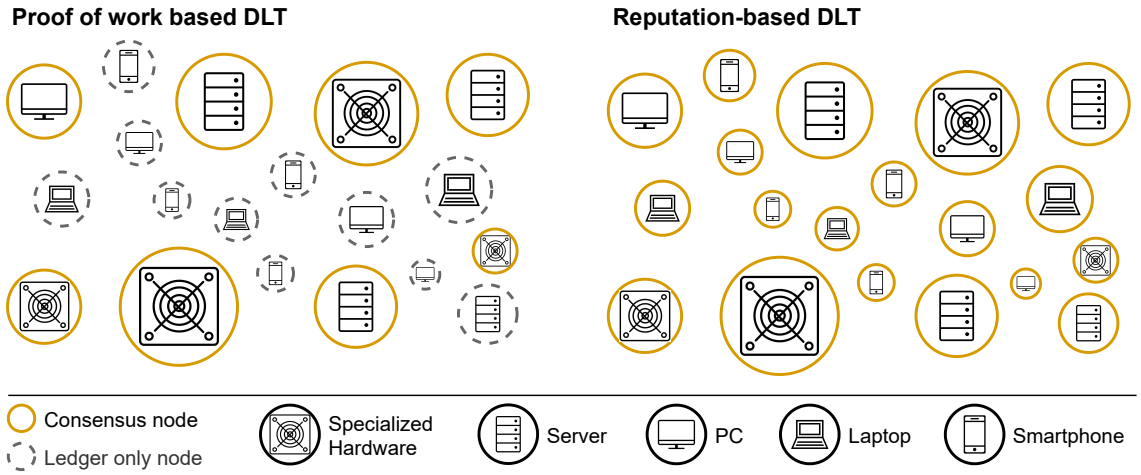

Figure 1.1: Comparison of nodes participating in consensus and ledger only in proof of work based DLTs (left) and reputation-based DLTs (right). In proof of work based DLTs practically only high-end nodes and specialized hardware can participate in consensus whereas in reputation-based every node because participation in consensus is not dependent on processing capabilities. Connections between nodes are not pictured out of brevity. The size of nodes describes their processing capabilities.

This class of DLTs adds design complexity to the original Bitcoin's blockchain and faces several challenges which we describe and address throughout the thesis.

One of the primary challenges in the Bitcoin network is the centralization of power. Miners typically use specialized hardware to compute a cryptographic puzzle faster than other nodes and add messages into the blockchain. This mechanism creates a costly filter formed by an elitist network (Figure 1.1 (left)) (13). Conversely, the permissionless nature of the DLTs considered in this thesis allows nodes with widely *heterogeneous* capabilities to participate in consensus and message generation (Figure 1.1 (right)). Such heterogeneity can be manifold:

- Bandwidth, latency, availability, and processing capabilities can vary between multiple orders of magnitude, as in traditional peer-to-peer networks (22).

- Unsteady processing rates of a node due to competing tenant applications and performance variability, especially in the public cloud environment (2, 23, 24).

- Protocols, geographical locations, node freshness, and software versions can differ widely in DLTs (3).

1.2 Challenges of Permissionless Reputation-based DLTs

Heterogeneity is a key feature of permissionless reputation-based DLTs but also introduces multiple challenges (10, 21). First, *who can vote?* In order to start the voting procedure, at least 51% of nodes must have received the most recent messages necessary to construct and verify the ledger state. In reputation-based DLTs, a score (reputation) is assigned to each node to determine nodes' reserved throughput shares and weights used during voting. Many DLTs assign reputation by linking it to a constrained resource, such as stake (e.g., proof of stake (PoS) DLTs (17, 25) or IOTA's Mana (10)); more sophisticated technologies try to evaluate whether nodes are well-behaving and contributing to the security of the network (19). A good reputation system should prevent Sybil attacks, where colluding nodes can gain disproportionate influence to manipulate the ledger state.

Second, unlike in Bitcoin, reputation-based DLTs require an explicit *distributed flow-control mechanism.* If powerful nodes issue new messages too fast without any flow control, then only high-end nodes will be able to keep up with the message processing and with the latest ledger updates. Hence, only few nodes, the ones with an updated ledger, can vote, thus increasing undesired DLT centralization.

Lastly, *maintaining maximum decentralization with high performance.* In DLTs, nodes are required to process all messages generated. Hence, to avoid any loss of synchronization or message drop, the network must operate at the speed of the slowest node, which can lead to low throughput. An additional challenge is given by the fact that the node's processing speed is not static as it varies based on the operating environment and performance fluctuations (2, 23, 24).

Though efforts have been put to tackle the issue of voting (10, 17, 25), limited attention has been paid to tackle decentralization and performance due to heterogeneity in DLTs. Such heterogeneity-related challenges are reminiscing of the early 2000s research in P2P content distribution systems (22, 26, 27, 28, 29), but the need for quality of service, and node reachability requirements sets modern DLTs apart from their P2P predecessors. We take inspiration from these works and recent networking research (30), and make a case for a *dynamic flow-control* protocol to react to the changing heterogeneity (i.e., computational capabilities) for maximizing throughput without sacrificing DLT decentralization.

1.3 Research Questions and Methodologies

Throughout this thesis we are discussing three main research questions regarding dynamic flow control in the context of heterogeneity in permissionless reputation-based DLTs. To answer the questions we make use of the following methodologies:

- **(M1)** Design, abstraction, prototyping (31, 32, 33)
- **(M2)** Experimental research, designing appropriate micro and workload-level benchmarks, quantifying a running system prototype (34, 35, 36)
- **(M3)** Open-science, open-source software, community building, peer-reviewed scientific publications, reproducible experiments (37, 38, 39, 40)

1. How to derive a right design for a dynamic, distributed flow-control protocol?

Flow control in the permissionless DLT setting builds on many diverse topics from networking research such as P2P systems, network security, congestion control, and flow control in traditional as well as distributed networks. Therefore, it is fundamental to understand these technologies (Chapter 2) and related work (Chapter 5) in-depth. Inspired by already existing approaches a new protocol can be iteratively derived **(M1)**.

2. What are the key challenges in designing a dynamic, distributed flow-control protocol in the permissionless DLT setting?

Finding the key challenges in any system is non-trivial as there are many challenges especially in DLTs. In this thesis, we are establishing the key insights through prototyping and designing targeted experiments as explained in Chapter 3 and Chapter 4 **(M1, M2)**.

3. How efficient is such a dynamic flow-control protocol for permissionless reputation-based DLTs?

Quantifying a system's performance first requires an objective set of metrics (e.g., in terms of latency and throughput). Additionally, in the context of permissionless DLTs, we need to measure the degree of (de)centralization to establish the egalitarian nature of the network. Then, the proposed solution can be compared against a baseline scenario to show improvements which are discussed in Chapter 4 **(M2, M3)**.

1.4 Thesis Contributions

In this thesis, we present Healthor[1], a novel heterogeneity-aware, lightweight flow-control mechanism for permissionless reputation-based DLT networks. Healthor captures heterogeneity by defining a node's *health* as a function of its processing power and the current network activity. The health updates of neighboring nodes are then used to calculate the message forwarding rates, thus dynamically adjusting the flow control per node. This basic mechanism allows high-end nodes to buffer messages for unhealthy nodes, thus protecting weaker nodes from being overwhelmed with wasted processing and rapidly adapting network load and bursts. With such a flow-control design more nodes are able to keep up with ledger updates and participate in DLT consensus, therefore increasing decentralization and network performance. Specifically, contributions of this thesis are as following:

1. We identify challenges (high centralization, poor performance, security) due to lack of heterogeneity-aware flow control in permissionless reputation-based DLTs (Section 2.4).

2. We propose Healthor, a lightweight distributed flow-control mechanism that leverages a node's health as a proxy of its heterogeneity and processing capabilities. We present its design choices, implementation, and tradeoffs. In comparison to other DLTs, which use leader election or fixed computation, we are among the first to introduce networking concepts and optimizations to the DLT space (Chapter 3).

3. We evaluate Healthor in OMNeT++ simulations for up to 5,000 nodes. Our results demonstrate that Healthor increases the degree of decentralization by 78%, improves throughput by 23%, and 95 percentile message latency by 76% while staying resilient against attacks (Chapter 4).

4. The preliminary idea of the thesis with initial results were published at a workshop: *Jonas Theis, Luigi Vigneri, Lin Wang, and Animesh Trivedi. 2020. Healthor: Protecting the Weak in Heterogeneous DLTs with Health-aware Flow Control. In Proceedings of the 4th Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers* (41).
   At the time of writing, an extended paper version is under review at a top conference.

[1] Union of the word *health* and the Germanic god *Thor* who is amongst other things associated with great strength and the protection of mankind.

5. Healthor's source code and simulation configurations are available on GitHub at `https://github.com/jonastheis/healthor`. Appendix 7.1 details how to compile, run simulations and reproduce experiment results with Healthor either from scratch or with the simulation result data set provided at `https://zenodo.org/record/4573698`. We follow the state-of-the-practice reproducibility instructions provided by the cTuning Foundation (42, 43), which is collaborating with ACM on a common methodology, reproducibility checklist and tools.

Plagiarism Declaration

I confirm that this thesis work is my own work, is not copied from any other source (person, Internet, or machine), and has not been submitted elsewhere for assessment.

Chapter 2

Background

In this chapter, we cover the fundamental concepts needed to follow this thesis. With Healthor, we discuss a distributed, dynamic flow-control mechanism for DLTs which involves a broad range of networking topics starting from traditional network concepts to early P2P systems and network and system security. Therefore, first congestion-control and flow-control concepts of traditional networks are explained. Second, we dive into P2P systems, their overlay networks' architecture, and some of their challenges. Then, basic concepts of DLTs are outlined, and we observe similarities with traditional networks and P2P systems. Finally, at the end of this chapter, we are fully equipped to understand the challenges and opportunities of DLT heterogeneity.

2.1 Traditional Networks

Any networked system — local area network and wide area network — relies on physical infrastructure to interconnect nodes. Heterogeneous hardware renders too much dependence on the underlying hardware impractical and thus pushes functionality to end hosts (44), especially in wide area networks and the Internet. Therefore, most communication protocols need to implement some form of traffic control in order to avoid overloading the network, i.e., congestion control (Section 2.1.1), and receiving nodes, i.e., flow control (Section 2.1.2) (45).

2.1.1 Congestion Control

Congestion control is concerned with regulating the network traffic as a whole. As such it prevents the network from becoming congested, i.e., being overloaded, by adapting sending rates so that fewer packets enter the network. Generally, a signal (explicit or implicit) is

used to detect congestion, and much of the efficacy of the mechanism depends on it. Congestion signals can be hardware-assisted (30, 46) or purely end-to-end (45, 47, 48, 49).

For example, in the case of TCP, the signal is packet loss: The algorithm slowly increases the sending rate (additive increase) to increase utilization. Once a loss event occurs, the sending rate is significantly lowered (multiplicative decrease) to avoid congestion (45).

Today's Internet's congestion control mostly consists out of one or another flavor of TCP (47). Newer, recently emerging protocols like BBR (49) and Copa (48), however, suggest that a loss-based signal is suboptimal with today's hardware and does not actually avoid congestion but only fills up buffers and eventually detects packet loss. Therefore, these protocols use other signals such as delay and congestion to improve throughput and latency (48, 49).

2.1.2 Flow Control

In contrast to congestion control, flow control (also called backpressure in distributed systems context) governs the transmission rate between two nodes, sender and receiver, making sure that the receiver is not overwhelmed by the sender. As with congestion control, the signal to regulate flows is paramount, and can be explicit or implicit as well as hardware-assisted or purely end-to-end (50, 51).

Flow control can broadly be divided into *credit-based* and *rate-based* flow control. The former being more frequently employed, e.g., in TCP, where a sliding window mechanism is used to determine the packets in flight (45). Rate-based flow control incorporates feedback (explicit or implicit) by the receiver to directly adjust the rate. This can be either *differential* (speed up/slow down) or *absolute* (new rate = value) (30, 50, 51).

2.2 P2P systems

P2P systems are a specific type of distributed system where interconnected peers (also called nodes) cooperate and share resources such as storage, CPU cycles, and bandwidth without a centralized entity. As such, the P2P design philosophy pushes all of its functionality to the edge of the Internet, mostly to end hosts, i.e., the peers (44, 52, 53). P2P systems have been popularized in the early 2000s (54), mainly through content distribution networks (27) such as Napster (55), Gnutella (56), KaZaa (28), and BitTorrent (57).

The following subsections explain basic concepts of P2P computing starting from the overlay network and its different architectures (Section 2.2.1) to incentives (Section 2.2.2)

and challenges such as heterogeneity (Section 2.2.3) and scalability (Section 2.2.4). Specifically, we focus on content distribution networks as these types of networks share many similarities with DLTs' underlying P2P overlay networks. The concepts and challenges of P2P content distribution networks covered in this section resemble those of DLTs nowadays (Section 2.3).

2.2.1 Overlay Network

P2P systems depend on a network of interconnected computers formed independently of physical infrastructure which is referred to as *overlay network* managed at the application level. Typically, overlay networks leverage standard internet protocols (TCP, UDP) for peer communication. Often P2P networks utilize epidemic broadcast to disseminate information throughout the network to all peers, i.e., a peer forwards messages to all or a subset of its neighbors. Overlay networks can be grouped by degree of centralization and structure, as detailed following (27).

Degree of centralization

Ideally, every node in a P2P overlay network is equally important and performs the same tasks. In practice, however, this is not always possible, e.g., due to heterogeneity, legislation, or system design, and different degrees of centralization can be observed.

Fully decentralized. The network is totally distributed, all nodes act simultaneously as server and client, performing the same tasks without any centralized coordination or component. These types of networks are censorship resistant and do not have a single point of failure. They suffer, however, from scalability issues and inefficiencies due to the fact that every node acts as server and client. Thus, every action, e.g., search needs to be executed in a distributed way (27, 52).

Partially centralized. Similar to fully decentralized, in partially centralized overlay networks nodes act as servers and clients at the same time. However, some nodes are dynamically selected — then called supernodes — to perform different actions, e.g., based on capabilities. Generally, these supernodes are selected in a distributed manner, and if a supernode fails the network topology adapts and selects another node. As fully decentralized systems, these systems promise censorship resistance — though slightly weakened due to more centralization — and do not have a single point of failure while targeting the aforementioned scalability issues of fully decentralized systems (27, 52).

Hybrid decentralized. As the name suggests, these systems are a hybrid between centralized and decentralized. Usually, a central server facilitates connections between peers

based on desired resources and capabilities. Only then do peers directly communicate with each other. Evidently, these systems suffer from a single point of failure and are vulnerable to censorship. On the other hand, a centralized server serves as a single source of truth and can increase search speed and accuracy (27, 52).

Structure

P2P networks can be *structured* and *unstructured* based on how nodes are connected to each other.

Unstructured. In an unstructured overlay network nodes randomly peer with each other without any global rules imposed on them. This makes joining and constructing a network easy, and the network as a whole resistant against churn, i.e., many nodes joining and leaving the network frequently. However, due to the lack of structure, these networks often suffer from scalability issues, e.g., a search query needs to be flooded through the whole network (27, 52, 58). Examples: Gnutella (26, 59), KaZaA (28), Napster (55).

Structured. Structured overlay networks mainly developed to address the scalability shortcomings of unstructured overlay networks. Hence, the topology is controlled via global rules to create a mapping between content and its location, e.g., node identity. Via distributed routing tables, e.g., in the form of a distributed hash table (60), queries can be efficiently routed to content providers. However, these types of overlays create overhead and are difficult to maintain when there is node churn (27, 52, 58). Examples: Chord (61), Kademlia (60), Pastry (62), CAN (63).

2.2.2 Incentives

One of the key problems of P2P systems is the lack of cooperation (free-riding), and providing proper incentives for users to adhere to the protocol (22, 29, 55, 64). As shown in (22) users tend to intentionally misreport information, are selfish, and exploit other users if there are no sufficient incentives to do otherwise. Many game-theoretic solutions, e.g. based on tit-for-tat (65), deeply ingrained into the protocol have been proposed to increase cooperation between peers and thus overall robustness, and efficiency of P2P networks (29, 66, 67). Other approaches suggest the use of sophisticated reputation systems to provide incentives to cooperate (68, 69).

2.2.3 Heterogeneity

Many studies have been conducted measuring public P2P networks in terms of topology, network traffic, node behavior, and node capabilities (22, 54, 56, 64). In their landmark

measurement study (22) Saroiu et al. show that P2P systems such as Gnutella and Napster are significantly heterogeneous. Bandwidth, latency, availability, degree of sharing, and processing capabilities can vary between three and five orders of magnitude. This implies that any P2P system needs to be careful when assigning special tasks to peers (e.g., supernodes), and that scalability problems can arise in fully decentralized systems where every node is supposed to perform the same tasks.

2.2.4 Scalability

As already in Section 2.2.1 discussed, unstructured P2P systems suffer from scalability issues which is why structured P2P systems emerged. These systems, however, experience high overheads if there is much node churn. Alternative approaches adjusting the network topology with supernodes have been proposed to get the best out of both worlds (26, 28, 56, 59). In (26) a scaling approach for the Gnutella network is proposed that leverages heterogeneity to dynamically adjust the network topology and puts high capacity nodes within short reach of most other nodes. Additionally, an active credit-based flow-control mechanism based on available capacity is used to avoid overloaded hot-spots, i.e., nodes hand out tokens to their neighbors where each token represents the willingness to accept a query.

2.3 Distributed Ledger Technologies

In 2008 the inconspicuous Bitcoin paper (4) written by the anonymous entity Satoshi Nakamoto sparked a revolution and ignited *distributed ledger technologies*. Though the underlying technologies were not novel (70, 71, 72, 73, 74), Bitcoin combined them in an ingenious way and created something that had been deemed impossible: consensus of replicated, shared, and synchronized data without a central entity in a permissionless and trustless setting where anyone can join and participate (4, 9). Since then many flavors of DLTs have emerged, not only to enable monetary transactions but more so to enable trustless code execution and thus paving the way for many more use cases and a distributed, trustless Internet, enabling parties to interact without trusting anyone (7, 75). DLTs broadly can be distinguished into permissioned, i.e., a central authority grants permission to participants, or permissionless, i.e., open access to anyone where participants do not know each other but cooperate through game-theoretical incentives (76). The latter systems pose more challenges due to their open nature. In this thesis, we focus on permissionless DLTs.

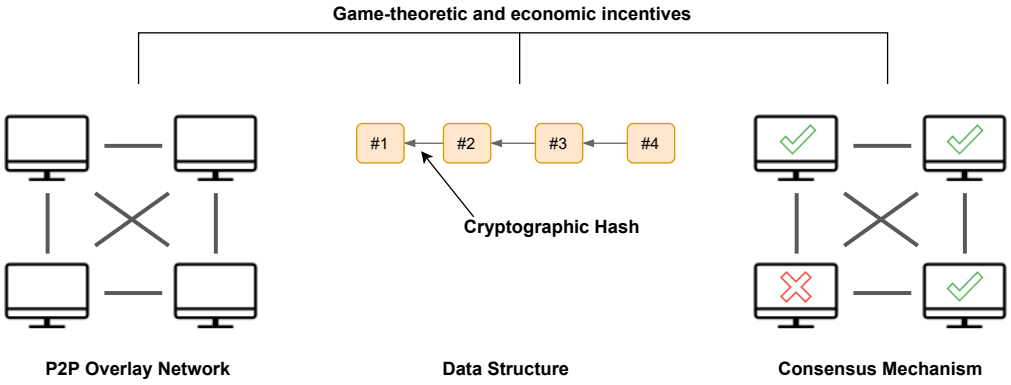

Figure 2.1: Permissionless DLTs comprise three main components: a P2P overlay network (left), an immutable data structure (center), and a consensus mechanism (right) are deeply fused via game-theoretic and economic incentives.

Generally, a DLT integrates three main components joining them together via its protocol and deeply ingrained game-theoretic and economic incentives as depicted in Figure 2.1. First, a P2P overlay network is utilized to disseminate state updates (Section 2.3.1). Second, every node keeps track of a shared, replicated, and immutable data structure (also called ledger) which is based on cryptographic primitives (Section 2.3.2). Lastly, nodes use a consensus mechanism to agree on a state in a distributed manner (Section 2.3.3).

Permissionless DLTs face similar challenges as P2P content distribution networks such as providing proper incentives (Section 2.3.4), fair access (Section 2.3.5) and scalability (Section 2.3.6).

2.3.1 P2P Overlay Network

Most DLTs build an unstructured decentralized P2P overlay network, either with manual peering, i.e., node owner are required to exchange connection details, or some form of automatic peer discovery and peer selection. Ledger state updates are disseminated in the form of messages (also called transactions) via epidemic broadcast so that every node eventually receives all ledger state updates (8, 9, 10).

In theory, all nodes have the possibility to exert the same functionality, thus making the network fully decentralized. However, this is not always the case in practice. While the overlay network topology might seem fully decentralized, multiple nodes might have the same owner or are physically colocated. In both cases the degree of centralization is increased (3, 12, 77).

To get an estimation of the level of centralization, we analyzed recently published literature on DLT deployments and decentralization (3, 12, 77). Our analysis shows that

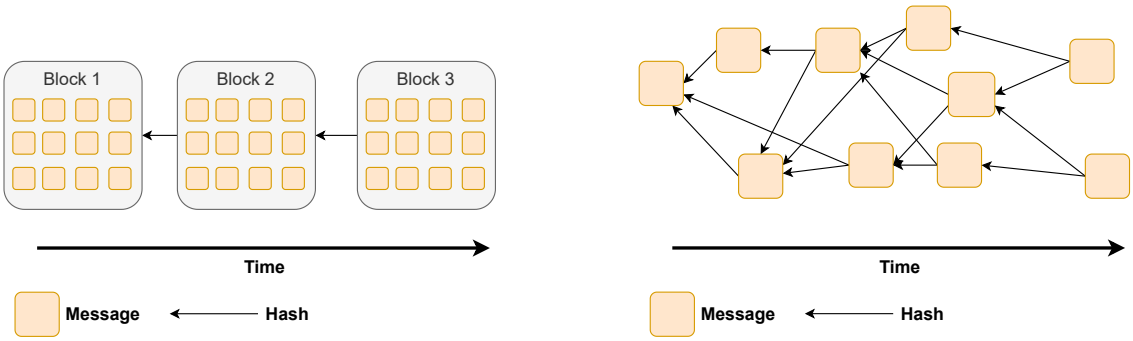

Figure 2.2: Data structures in DLTs. A blockchain groups messages into blocks and links blocks together (left). The Tangle consists of messages that reference 2 other messages (right), thus forming a DAG.

(Table 2.1) a significant amount of nodes in DLT networks run on only few big cloud hosting providers. For example, 62% of publicly reachable Ethereum nodes are running on cloud hosting providers. In the IOTA network this is even more extreme with 69% of publicly reachable nodes running in the cloud.

2.3.2 Data Structure

The ledger state in a DLT is derived from an immutable data structure, that can be compared with an append-only log. Every node in the network has a copy of the ledger state and can thus verify the validity of new updates locally. Newly joining nodes can bootstrap by simply downloading the ledger from their neighbors, and then make sure that the ledger state is valid locally (5, 7, 9, 20, 21).

Blockchain. A blockchain is a linked list of blocks where each new block contains a cryptographic hash of its predecessor's content as shown in Figure 2.2 (left). A block mainly consists of a number of state updates, e.g., in the case of Bitcoin monetary transactions. This essentially creates an immutable chain of blocks where any change to a block invalidates all future blocks as well. The longer the chain, the harder it is to change any content because all future blocks would be invalidated, hence it is tamperproof. A blockchain is totally ordered: blocks are issued, e.g., with consensus on the longest chain, at regular intervals and state updates within a block are deterministic (5, 7, 9).

Directed Acyclic Graph (DAG). A directed acyclic graph (DAG) is a graph without directed cycles, i.e., it grows in one direction. IOTA's Tangle (20) is a DAG where messages are linked together via their cryptographic hashes instead of being grouped into blocks. Figure 2.2 (right) shows this data structure. Similar to a blockchain, linking messages

Name	Date	Size	Hosting	Top Providers
Bitcoin (Bitnodes (78))	21/11/2020	11,122	4,195 (38%)	Tor Network: 2,827 (25%)† Hetzner: 1,049 (9%) Amazon: 803 (7%) OVH: 490 (4%) DigitalOcean: 455 (4%) Google: 360 (3%)
Bitcoin (Mariem et al. (77))	07/05/2019	9,476	6,159 (65%)	Hetzner: 1,042 (11%) Amazon: 805 (8.5%) DigitalOcean: 616 (6.5%) OVH: 550 (5.8%) Comcast: 351 (3.7%)
Ethereum (ethernodes.org (79))	21/11/2020	9,517	5,855 (62%)	Amazon: 1,778 (19%) Alibaba: 1,106 (12%) Hetzner: 541 (6%) Google: 385 (4%) DigitalOcean: 326 (3%)
Ethereum (Kim et al. (3))	08/05/2018	8,309	3,722 (44.8%)∗	Amazon Alibaba DigitalOcean OVH Hetzner Google
IOTA (thetangle.org (80))	21/11/2020	302	202 (69%)	Contabo: 66 (22%) Hetzner: 63 (21%) Netcup: 37 (12%) Amazon: 10 (3%) DigitalOcean: 6 (2%)

Table 2.1: Network size (publicly reachable) and distribution of nodes running on cloud hosting providers.
∗ Only top 8 ASes, no exact values published in (3).
† Not included in *Hosting* due to unknown service provider.

together via their cryptographic hashes makes the data structure immutable and tamper-proof. The Tangle is not totally ordered as messages can be attached simultaneously by multiple users which promises better scalability compared to blockchains. However, there is added complexity for nodes to verify the ledger state and come to consensus. For example, before a message can be verified it needs to be solid, i.e., its entire history needs to be known to the node. In case a node is missing a message it can ask its neighbors via *solidification request* (7, 20).

2.3.3 Consensus Mechanism

The consensus mechanism is at the core of every permissionless DLT. It is a set of rules that combines the P2P overlay network, data structure as well as some form of leader election, e.g. to select a block producer, and (virtual) voting with game-theoretical incentives and bakes the results into the immutable ledger. In this way, the data structure does not only serve as an immutable ledger database but also as a verifiable instrument of consensus. It enables a Byzantine Fault Tolerant P2P network of anonymous nodes that are free to join and leave at will (81, 82).

Leader election

Proof of Work (PoW). PoW was initially invented as an email spam protection mechanism. The principle is fairly straightforward. A probabilistic puzzle needs to be solved usually producing a nonce (a non-negative integer) that is hard to find but easy to verify, thus a scarce resource — in this case computing power and energy — needs to be invested (74).

In DLTs, specifically first with Bitcoin, PoW is used as a leader election mechanism. The first node to solve the puzzle is able to produce a block and gets rewarded. In Bitcoin, it works as follows: So-called miners (special types of nodes) take transactions and combine them into a block's essence. Additionally, the block contains a reference in form of the cryptographic hash to the previous block. Then a miner needs to find a cryptographic hash of the current block's content such that the block hash starts with a specified number of 0s. Specifically, this means counting up the nonce and generating — via trial and error, hence probabilistic — a valid block hash. The first miner to find a valid block broadcasts it to the network (9, 81, 82).

Reputation-based. Reputation-based leader election works similar to leader election with PoW in the sense that a scarce resource (e.g., reputation or stake) is necessary to become the leader and produce a block. In contrast to PoW, no extensive calculation

is necessary but instead, for example in the case of PoS, a validator (similar to a miner in PoW) is selected weighted by its (delegated) stake. The selected validator can then take transactions, chain it together with the previous block and broadcast it into the network (81, 82).

(Virtual) Voting

Longest chain rule. A block producer implicitly casts a vote for all previous blocks by appending its block to the chain. This is known as virtual voting as the vote is implicitly cast. Generally, everyone follows the longest chain because it has the highest amount of scarce resource accumulated (remember that consensus is encoded in the data structure). For example, in PoW this means that an attacker needs on average more than 51% of the hashing power of the entire network to produce a longer chain than the honest part of the network. Only then can the attacker fool the entire network (81, 82).

Direct voting. In some reputation-based DLTs that do not employ a blockchain as underlying data structure such as IOTA (10) and Nano (83) nodes are queried to cast a vote. The weight of a vote is usually scaled by reputation.

2.3.4 Incentives

Game-theoretic and economic incentives are deeply ingrained into the consensus protocol to resolve problems that plagued earlier permissionless P2P systems (84). In Bitcoin, miners not only receive a reward for creating a block but also receive the so-called transaction fee, i.e., the fee a user needs to pay to include their updates into a block. Only the miner whose block is accepted (usually the first) gets the rewards, effectively creating sunk cost for all others. In turn, the probability for a miner to be first increases the more computing power and the longer it keeps mining and thus increases overall network security (4, 9).

However, this very incentive structure leads to centralization in practice. First, specialized hardware is more efficient at solving the puzzle, creating advantages for these usually non-consumer friendly devices (13). Second, mining pools get together and share the computational expense and subsequently potential rewards. Few mining pools constitute a majority of the network's hash rate, at worst putting the consensus at risk and endangering censorship resistance (9).

Reputation-based DLTs utilizing PoS incentivize validators to act honestly similarly. Though, instead of creating mining races, validators are encouraged to be honest because their stake (monetary value) is at risk (17, 19, 25). Feeless reputation-based DLTs such as IOTA (10) and Nano (83) often struggle to provide hardened incentives because there is

no monetary reward for participation in the network. Here the only incentive to be honest is the exclusion of the protocol.

2.3.5 Fair Access

As discussed in Section 2.3.4, in many DLTs a user needs to pay a small fee so that a block producer includes the update into a block. While this fee is part of the intrinsic incentives of consensus (rewarding the block producer), it also creates an access and spam filter to the network. Block producers can decide which updates to pick, but generally the one's paying most get first included in a block. Therefore, a spam attack on the network is costly. However, in times of high network load fees can grow extremely high and render the network practically unusable for many use cases (9).

Specifically, in the case of micro-payments in the machine-to-machine economy fees are often bigger than the transacted value itself. Feeless DLTs such as IOTA (10) and Nano (83) do not have fees nor inflation. This creates the need to regulate network access and a spam protection mechanism (21). IOTA's Mana (10) (a measure of reputation) in combination with a distributed congestion control (85) solves the problem for IOTA's DLT.

2.3.6 Scalability

Popular DLTs nowadays suffer from poor scalability in terms of message confirmation latency and throughput, e.g., the Bitcoin network is able to process a maximum of 10 messages per second (MPS) while blocks are only produced every 10 minutes (9). This means that confirmation latency is at best the time to be included in the next block but at worst, e.g., when the network utilization is high, it can happen that a transaction is not confirmed for multiple blocks. This scaling problem exacerbates the previously mentioned fees as users start a bidding competition (Section 2.3.5).

Efficient scaling of DLTs (also known as sharding) is, therefore, a major concern and topic of a host of research, most of which focuses on the inherent scalability issues of DLT consensus (9, 14, 15, 16, 19, 86, 87, 88). Since the consensus layer does not yet support more throughput, little has been done in terms of optimizing the networking layer (86).

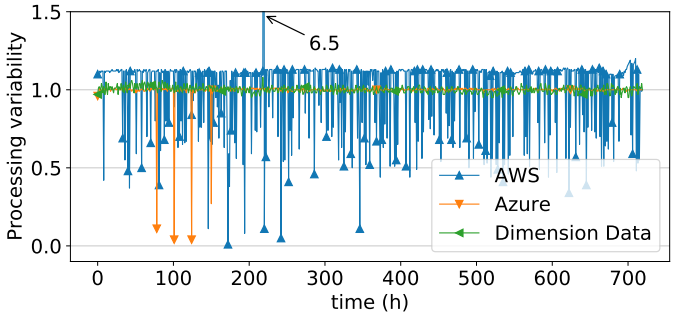

Figure 2.3: Processing variability in cloud providers. Data from (2) measured hourly over a period of 30 days. Normalized (mean=1) to show relative performance variability on a single cloud provider.

2.4 Putting It All Together: Challenges and Opportunities of Heterogeneity in DLTs

The previous sections covered the basics of congestion and flow control in traditional networks, introduced P2P networks, and explained DLT fundamentals. With this basic working model in mind, we now discuss what is the extent of heterogeneity in contemporary DLT networks, what happens if heterogeneity is ignored, and what opportunities present if we can leverage it.

The Nature of DLT Heterogeneity

Recall the analysis of DLT network centralization (Section 2.3.1), and that at least 62% of publicly reachable Ethereum nodes run on only few big cloud hosting providers. In the IOTA network, this is even more extreme with 69% of publicly reachable nodes running in the cloud.

Though on the surface cloud hosting seems to offer a more homogeneous environment, it is not the case in practice. First, cloud providers offer a bewildering array of choices in terms of configurations, capabilities, and cost of systems resources like virtual machines (VMs), which has lead to a series of work in workload optimizations for heterogeneous cloud resources (89, 90, 91, 92). Such heterogeneous choices imply that there is no single ideal VM that DLTs can choose to deploy. Moreover, even with the choice of a VM, there is significant performance fluctuation over time (2, 23, 24, 93). For example, we took cloud performance traces from (2), which has collected the performance of CPU-intensive benchmarks over a period of 30 days from three cloud providers (AWS, Azure, and Dimension Data), and normalized the performance to the mean value in the performance. We plot the performance variation time series (on the y-axis) with respect to time (on

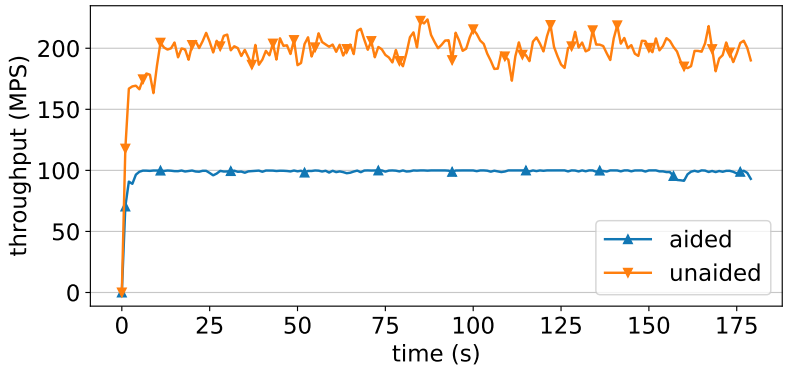

Figure 2.4: Mean throughput (higher is better)

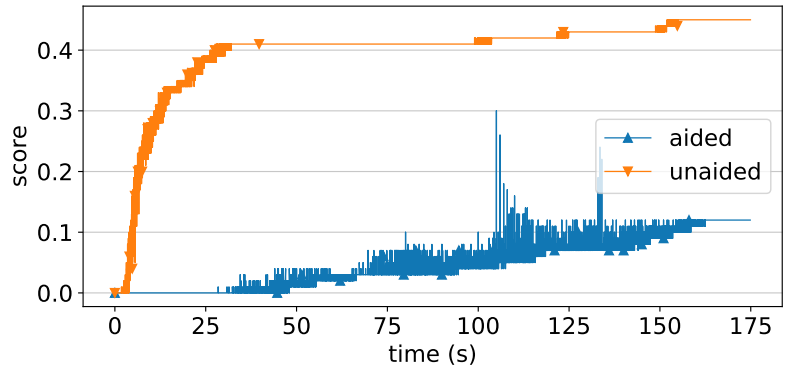

Figure 2.5: Centralization value (lower is better)

Figure 2.6: Comparison of aided and unaided heterogeneity in voting-based DLTs in a heterogeneous network with 100 nodes. (a) Mean throughput (processed messages) of all nodes. Higher is better. (b) Centralization value (Section 4.2). Lower score indicates higher degree of decentralization (desired).

the x-axis) in Figure 2.3. In that way, we can easily observe the performance variability over time for a single VM of one of the cloud hosting providers. The key finding from the figure is that there exists more than an order of magnitude performance variability in hosted cloud providers. It is this variability that leads to processing heterogeneity even for cloud-hosted DLTs.

What happens if we disregard heterogeneity in DLT processing?

Disregarding heterogeneity leads to centralization of voting power. To quantify the impact of heterogeneity on centralization, we run an experiment with 100 nodes in OMNeT++. In the experiment, the incoming rate of new messages is set to 200 messages per second (MPS). The 100 nodes are modeled with a mean processing rate between 90-350 MPS with

their message processing rate modeled after the traces from Figure 2.3 where 25% of nodes are on AWS, 15% on Azure, and 10% on Dimension Data[1]. During the experiment, we measure a metric called *centralization value*, which is calculated as a ratio of nodes which are left behind and can not vote (due to their inability to process high rates of new ledger update messages) and the total number of nodes (Section 4.2 for details). Hence, the lower the score, the higher the participation in voting, thus, higher DLT decentralization (desired). We further investigate two network scenarios: aided and unaided. In the aided setup, a DLT can enforce a fixed throughput rate (i.e., flow control) which is calculated keeping the slowest node(s) in mind (e.g., with a minimum DLT joining requirement), thus ensuring a certain level of decentralization at the expense of resource utilization. The current IOTA Coordicide solution proposes this (10). In the unaided setup there is no network-level support for heterogeneity.

Figure 2.6 shows our results for aided and unaided cases. First, we look at the throughput (the y-axis) with time (on the x-axis) of both cases as shown in 2.4. As expected, the aided case leads to a stable throughput of 100 MPS while underutilizing the remaining processing capability of the network. In comparison, in the unaided case the throughput increases until around 200 MPS (the orange line), thus proving that the network has spare, underutilized capacity. However, when we analyze the centralization values of aided and unaided configurations we observe an opposite picture (Figure 2.5). As the unaided configuration delivers higher performance, it leaves a large chunk of slow nodes out of sync (up to 40% by the time 30 seconds), i.e., the incoming rate of new messages is greater than the processing capabilities of a node. These out-of-sync nodes can not participate in voting, thus, allowing only faster nodes to have full control of the consensus procedure. In contrast, the fixed-rate aided DLT only leaves less than 10% nodes out of sync, though at the cost of a poor, underutilized DLT network.

However, both previously shown scenarios are not optimal as they either statically trade throughput for decentralization or decentralization for throughput. A desirable solution should deliver high throughput at all times while guaranteeing a high degree of decentralization, and this is exactly what we propose with our Healthor.

[1]For more details of how these values are calculated and modeled please see Section 4.1.

Chapter 3

Design of Healthor

Healthor is a distributed flow-control protocol to improve the decentralization and performance in a heterogeneous DLT network. Before we introduce the details of the Healthor protocol, we first briefly present the network model and our assumptions in the following section.

3.1 Network Model and Assumptions

We denote the set of all nodes participating in the network as $\mathcal{M}$, where each node $m \in \mathcal{M}$ has a set of inbound and outbound neighbors denoted by $\mathcal{N}_m \subset \mathcal{M}$. Figure 3.1 shows a node model and its neighbors. A node and each of its neighbors are connected via bidirectional channels over which they exchange *messages*. A message contains data, e.g., monetary transactions, and a hashed reference to two previous messages as its parents, effectively building an immutable directed acyclic graph like the Tangle (Section 2.3.2). Network membership management and bootstrapping of nodes are handled using an external service.

A node can perform various operations, namely issuing, receiving, processing, and forwarding messages. We assume that a node can issue new messages at a recommended

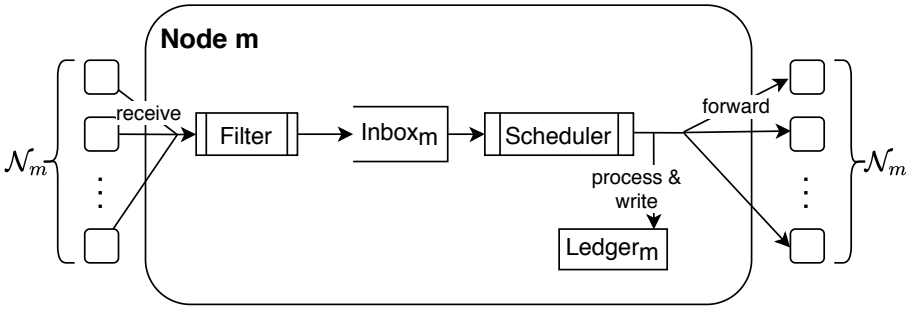

Figure 3.1: Model of a node m and its neighbors $\mathcal{N}_m$.

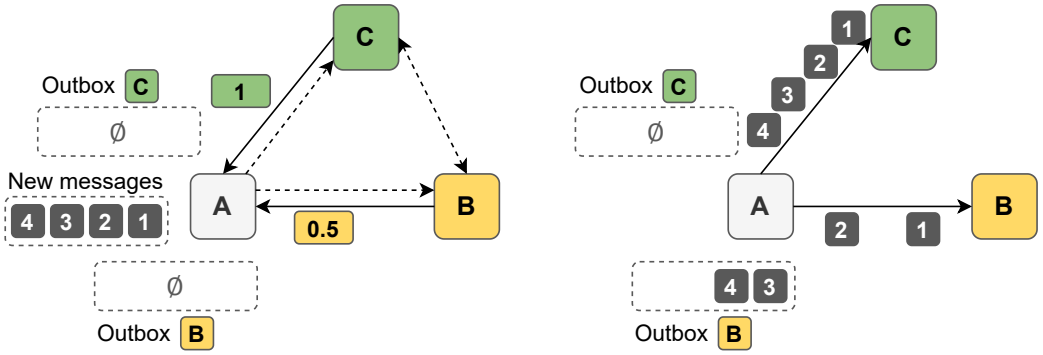

Figure 3.2: High-level design of Healthor. Nodes periodically send their health to their neighbors (left). Node A adjusts its rate according to a neighbor's health and buffers messages in an outbox per neighbor (right).

rate which is enforced by the network congestion mechanisms (85). In this work, we focus only on the end-host flow-control mechanism. On receipt of a message, the node filters out duplicates and pushes unseen messages to its inbox, a buffer with limited size. Based on the message's cryptographic signature, the node can identify if it has processed all parent messages for a new message, i.e., it is solid. In case of missing parent messages, it requests the missing messages from its neighbors via an implementation-specific synchronization mechanism like pull-action in Gossip-based networks (94). For any other message for which the node has all parent messages (i.e, entire history), the message is scheduled for processing which includes cryptographic signature verification, and then writing ledger updates to persistent storage. After processing, the message is forwarded to the neighbors.

We distinguish between two different operating modes of a network, *aided and unaided heterogeneity*. In the aided case, we assume that an overall processing rate ν_{net} (typically messages per second (MPS)) of the network is defined. This is the message rate at which the network as a whole should operate (we will discuss more about it in Section 4.4). Let ν_m be the variable message processing rate of a node m. Hence, node m would be able to process and forward messages at rate $\min(\nu_{net}, \nu_m)$. Ideally, $\nu_m = \nu_{net}$ at any time, meaning that a node m is able to operate at the overall network rate. However, operating conditions may lead to performance fluctuation, thus leading to accumulating messages in the inbox. In case the inbox runs full a tail drop policy is used, i.e., new messages that would cause the buffer to overflow are dropped. In the unaided case, a node m is free to forward messages at its rate ν_m. The current Healthor protocol design is for the aided case, but in Section 6.2.1 we discuss how we can relax knowing about ν_{net} and what implications it has.

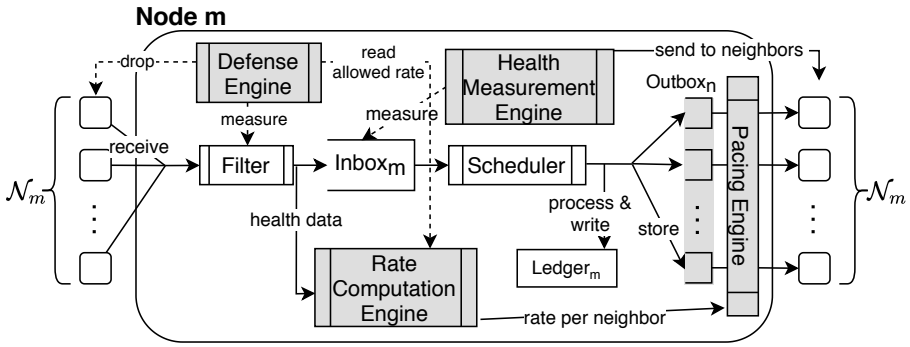

Figure 3.3: Model of a node m with Healthor. New components compared to Figure 3.1 are highlighted in gray.

3.2 Workings of Healthor

The basic idea behind the protocol is to rate-limit message forwarding in a DLT network based on a neighbor's current message processing capacity, termed as its *health*. Intuitively, the notion of health captures the dynamic heterogeneity of the DLT network, which might be changing over time. Figure 3.2 presents an example showing the intuition behind Healthor's design at a high level. The figure shows a DLT network with 3 nodes (A, B, and C). Node A has new messages to forward to nodes B and C. Before A calculates the forwarding rate, it receives the health updates (0.5 for B, 1.0 for C), and then calculates the rate based on the updates. Here in this example, node C gets all the messages, whereas B only gets half, while the other half is buffered by node A on the behalf of node B. This basic mechanism is the key insight in our flow-control protocol where more capable nodes can buffer messages to accommodate performance fluctuations in weaker nodes.

Healthor operates at the application layer and maintains connections to neighbors in a group communication setup. It is inefficient to rely on existing mechanisms such as TCP-based backpressure because (1) TCP-based backpressure runs independently of the application level and would need modifications to the TCP implementation which is not feasible in public DLTs; and (2) TCP is a point-to-point protocol which cannot efficiently handle group communication dynamics.

Figure 3.3 depicts an updated model of a node m with the new components from Healthor highlighted. Healthor is a framework encompassing four separate engines, inspired by TIMELY (30), that uses local queue building and health updates as a signal. Additionally, a node m has an $Outbox_n$ for every neighbor $n \in \mathcal{N}_m$ where references to messages for n are stored before forwarding. In the following section we introduce these new engines and associated design decisions.

3.3 Health Measurement Engine

The *Health Measurement Engine* is at the heart of the mechanism. By introducing the notion of health $h_m \in [0, \infty)$, a node m can express its fitness regarding processing the maximum number of messages as defined by the expected network's processing rate ν_{net} (in the aided case). A node periodically calculates its health based on its inbox occupation $l_m \stackrel{\text{def}}{=} len(Inbox_m)$ as following

$$h_m = \begin{cases} \frac{\nu_m}{\nu_{net}} & \text{if } l_m < \nu_{net}, \\ 1 - \frac{l_m}{l_{capacity}} & \text{otherwise.} \end{cases} \tag{3.1}$$

where $l_{capacity}$ is the maximum size of the inbox, configured on the startup of node m.

3.3.1 Choosing a Signal

It is not trivial to find a reliable signal for rate control in a distributed, permissionless DLT setting. Relying on special hardware support such as Explicit Congestion Notification or changes to the operating system stack are infeasible in a public DLT. Due to the ever-changing nature of available computing resources, an initial announcement of capabilities is also not practical. Therefore, the solution must come from the end-host's application level.

Since all messages are delivered to all nodes and every node keeps track of received messages in relation to ν_{net}, every node knows the current health level of the network. Therefore, the inbox occupation gives a reasonable assessment of *how much a node is in sync*, i.e., whether it is able to receive and process state updates in a consistent and timely manner. A low inbox occupation signals a node being able to process at the network's pace, being healthy. Conversely, a high inbox occupation conveys that it is struggling to keep up with network activities. Thus, the node is unhealthy.

3.3.2 Different Means to Compute Health

As *health* is such a central component of the mechanism, we experimented with several approaches before concluding on the one described in Eq. (3.1). Firstly, only using the second branch of Eq. (3.1) leads to $h_m \in [0, 1]$ and a node m always being slightly unhealthy as soon as there is any message in the inbox. Our initial simulation results showed consistently poorer performance than the ideal aided case (i.e., $\nu_m = \nu_{net}$) as the ideal case represents the achievable theoretical maximum. By introducing the condition $l_m < \nu_{net}$ a node can still be seen as healthy if its inbox is occupied with a few messages as long as they

remain less than the maximum messages expected at the network rate, the performance was equal to the aided case. Lastly, allowing a health rate greater than 1 enables nodes to temporarily go faster than ν_{net}, which we adopted as our final way to compute h_m.

3.3.3 Health Messages

A health message is very lightweight, containing simply the health of the node m as a double-precision floating-point number. It, therefore, is only 64 bits of data. Additionally, in a real system, like any message, it should contain a node signature to verify a valid origin of a message. If a health message is lost for any reason, a neighbor n simply continues forwarding at its last known rate for a node m until it receives a new health message.

3.4 Rate Computation Engine

On receipt of a neighbor n's health data the node's *Rate Computation Engine* calculates the message forwarding rate for this neighbor n as shown in

$$r_n = \nu_{net} \cdot h_n. \tag{3.2}$$

The node computes the forwarding rate r_n linearly according to the neighbor's health h_n. Therefore, it can even go faster than the target network rate ν_{net} if its neighbors can process messages at the forwarding rate of r_n. Nevertheless, a node can only forward as much as there is network activity (new messages are issued), and it is able to process itself.

3.5 Outboxes

As in Figure 3.3 illustrated, a node m has an $Outbox_n$ for every neighbor $n \in \mathcal{N}_m$. When a message is processed, a reference to it is added to every outbox. The original message is stored in the node's local ledger. The *Pacing Engine* takes care of forwarding messages to neighbors at their respective rates.

Drop Policy

Similar to the inbox, an outbox is a buffer of limited size, defined on node startup. In case an outbox runs full, a tail drop policy is used (Figure 3.4 top), i.e., new messages that would cause the buffer to overflow are dropped while message requests are prioritized as shown. We also experimented with random (Figure 3.4 bottom) and head drop (Figure 3.4 center) policies. However, these turned out to be unfavorable due to the fact that messages are generally forwarded in order. If messages from the beginning of the buffer are dropped,

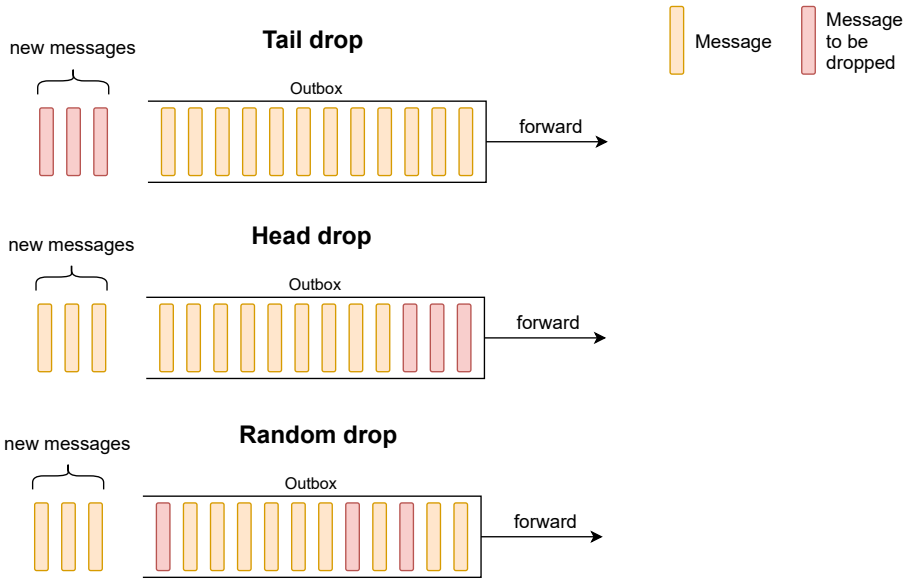

Figure 3.4: Drop policies of outboxes.

the receiver needs this already dropped message to process later messages. It, therefore, needs to request these dropped messages adding even more overhead.

3.6 Pacing Engine

The *Pacing Engine* fetches messages from an $Outbox_n$ for a neighbor n and forwards them at the rate calculated by the *Rate Computation Engine*. It essentially controls each flow of messages to every neighbor to achieve the given forwarding rate r_n. A possible implementation in a real-world system could make use of one thread per $Outbox_n$ that pulls messages from the outbox and forwards them while inserting delays to match r_n.

3.7 Defense Engine

It is essential to protect nodes against exploitation and adversarial behavior in a permissionless DLT setting. Therefore, the *Defense Engine* locally monitors a node's neighbors behavior and initiates appropriate actions if a protocol violation is suspected. Fundamentally, it provides incentives, hardens the flow-control mechanism, and makes nodes resilient against attacks. There are two main attack vectors on Healthor, namely exceeding the allowed forwarding rate and manipulation of health updates.

3.7.1 Exceeding Allowed Forwarding Rate

The basic idea is that, in the aided case, the expected network rate ν_{net} is known to all the nodes in the network. Hence, a node's neighbor n calculates the forwarding rate as defined in Eq. (3.2) and its pacing engine forwards out messages at this rate (i.e., the allowed rate). However, a neighbor may diverge from this rate because of being unhealthy itself, connection issues, or adversarial behavior. In any case, an extremely large divergence of the allowed rate cannot be tolerated. Therefore, every node m keeps track of the rates of every neighbor n by simply counting the received messages per neighbor. The *Defense Engine* of a node m periodically creates a moving average of the allowed rate within the time window tw as well as a moving average within the same window for every neighbor's receive rate. If a neighbor exceeds the allowed rate β-times in a row, the neighbor is dropped. Likewise, if a neighbor falls below the rate β-times.

3.7.2 Manipulation of Health Updates

A node m sends the same health update to all its neighbors. Therefore, health updates can be absent or manipulated by an adversary. If no health updates are received, a node m simply uses the known previous health of a neighbor n. At startup every node considers all its neighbors to be healthy, i.e., $h_n = 1, \forall n \in \mathcal{N}_m$. Generally, an adversary can only influence its own view on the network traffic by manipulating its health. For example, if she lies and sends different health messages to distinct neighbors, each neighbor will send at different rates according to the protocol. However, this does not have any influence on the neighbors.

On the other hand, an adversary could try to inflate outboxes of neighbors and to slow them down by pretending to be unhealthy. Inflation of outboxes, however, is not possible due to their limited size and drop policy. Nonetheless, a neighbor should not waste resources and therefore the *Defense Engine* implements a similar strategy to the forwarding rate. A node m keeps track of the health of every neighbor n by simply recording the outbox occupation. Recall that a node buffers messages for unhealthy neighbors. The unhealthier a neighbor n the higher the occupation will be. If a neighbor n is unhealthy for too long, $Outbox_n$ will run full, and eventually, the neighbor is dropped after β measurements.

3.8 Summary

In this chapter we first introduced our network model and assumptions (Section 3.1) and then detailed Healthor's design (Section 3.2) and its four engines, comprised of Health

Measurement Engine (Section 3.3), Rate Computation Engine (Section 3.4), Pacing Engine (Section 3.6) and Defense Engine (Section 3.7). Together with an outbox per neighbor (Section 3.5) these engines form an application-level flow-control mechanism for DLTs that uses health, a metric based on a neighbor's current message processing capacity, as a signal to rate control between peers. This signal is paramount for a dynamic, distributed flow control as it needs to be resistant to cheating, capture heterogeneity as well as ever-changing processing rates, and provide fair rates while utilizing available resources. In the next chapter, we analyze Healthor with its health signal in depth.

Chapter 4

Evaluation

We evaluate Healthor at two different scales. First, we explore *node properties* such as throughput, latency, and individual load at a small scale. These microbenchmarks are conducted simulating a network of 10 nodes such that tunable parameters of Healthor can be separately investigated. Second, we examine global *network properties*. In macrobenchmarks, we shift focus on the overall performance of the network and study decentralization, throughput, and tail latencies. Along with this evaluation, we discuss the following questions:

- *Does Healthor take load away from unhealthy nodes and allow nodes to stay (longer) in sync?* Our findings in Section 4.3 suggest that Healthor reduces load on low-end nodes and allows nodes to stay in sync when they could not with aided heterogeneity.
- *What is the influence of processing heterogeneity on decentralization and throughput?* We observe in Section 4.3 that some nodes that fulfill the network requirements nominally, i.e., ν_m is greater than ν_{net}, fall out of sync because of heterogeneity, thus increasing centralization.
- *Can Healthor improve decentralization, throughput and/or latency?* Our results in Section 4.4 demonstrate that, indeed, Healthor can improve all three properties.
- *Does Healthor provide the above improvements while staying resilient against attacks?* We show in Section 4.5 that nodes can detect protocol violations and protect themselves against attacks reliably.

4.1 Setup

We built a discrete-event simulator using OMNeT++ to simulate the permissionless, voting-based DLT modeled in Section 3.1 and test our mechanism. In our experiments,

we employ small-world networks of various sizes $|\mathcal{M}|$ where each node has between 2-4 random neighbors. Such random pairing is done in real-world applications that adopt a Kademlia-like (60) peering mechanism such as Ethereum (8), IPFS (95), BitTorrent (96), and Storj (97). The distance between two randomly chosen nodes is in the order of $\log|\mathcal{M}|$ (60). If a node gets out of sync, it goes offline and its neighbors repeer with other random nodes that have less than 4 neighbors.

Parameters

We adopt a Poisson process as the network processing rate $\nu_{net} = 100$ MPS for our experiments with aided heterogeneity and Healthor. Theoretically, ν_{net} is not defined for unaided. However, for simplicity, we assume $\nu_{net} = 200$ MPS for our experiments in this case. A random subset of nodes issues messages following a Poisson distribution with parameter λ_m, so that $\sum_m \lambda_m = \nu_{net}$. Channels are assumed to have a delay between 50ms and 150ms uniformly at random to simulate real network conditions (3, 12).

The variable processing rate ν_m of a node m is modeled according to the cloud performance traces shown in Figure 2.3. As a conservative approximation of real-world DLT P2P networks (3, 12, 77, 78, 79) (detailed in Section 2.3.1), we adopt a distribution of 50% constant, 25% AWS, 15% Azure, and 10% Dimension Data. Every ν_m is randomly scaled between $0.9\nu_{net}$ and $3.5\nu_{net}$ (in the case of unaided, the original ν_{net} value with which is compared is used). For example, if a node m is of type *AWS* with $1.5\nu_{net}$ its *mean* processing capabilities are $\bar{\nu}_m = 150$ MPS but vary over time as pictured in Figure 2.3 where the x-axis is randomly shifted.

Every node's *Health Measurement Engine* computes its health h_m and sends it to its neighbors every 1 second. Since we do not consider message priorities, a node's scheduler operates according to FIFO. A node's *Defense Engine* calculates allowed rates at an interval of 1 second and creates moving averages over a time window $tw = 3$ seconds. A neighbor is dropped after violating the protocol $\beta = 5$ times in a row.

4.2 Metrics

4.2.1 Centralization Score: Quantifying (De)centralization

In P2P networks, decentralization is the property of not relying on any centralized component. DLTs work, by definition, in a decentralized way. However, while in theory no centralized components are present, to prevent Sybil attacks nodes have different influences on consensus. Hence, if a node assumes too much power (e.g., hashing power in

Bitcoin (9), concentration on major cloud hosting providers (3, 12, 77)), we can conclude it has exceeding control of the network. Decentralization is a fundamental property of DLTs, and with the *centralization score* we introduce an easy way to compare the degree of decentralization in a voting-based DLT with the ratio of nodes being able to participate in voting.

The *centralization value* is the number of nodes that are not able to process all messages within a defined time window d normalized by the total number of nodes and is defined as

$$c_{value}(t) = \frac{\sum_{m \in \mathcal{M}} unsync_m(t)}{|\mathcal{M}|}, \tag{4.1}$$

where

$$unsync_m(t) = \begin{cases} 1 & \text{if } m \text{ processed all messages in } [t, t+d], \\ 0 & \text{otherwise.} \end{cases}$$

As such, a lower value is better because more nodes in the network are able to participate in consensus. In our evaluation we adopt a time window $d = 5$ seconds, i.e., a node is considered not being able to participate in consensus if at least one message has been received by the node later than 5 seconds from the time the message has been issued, also taking into account network delays. Considering that voting usually takes place in rounds it is reasonable to assume that a node being a bit behind is still able to participate. In the case of the IOTA Coordicide protocol voting rounds are initiated every 10 seconds (10). Recent PoS DLTs like Polkadot (19) adopt block times of around 5 seconds. Also in these systems, a node needs to be able to receive and process a transaction within this bounded time window in order to be able to produce or validate a block, thus taking part in consensus.

To show the evolution of centralization we plot the centralization value over time, which in simulation can be easily determined and in real world scenarios can be inferred by the voting participation of nodes. Accordingly, the *centralization score* is the mean of the centralization value over a given duration D and expresses the degree of decentralization in a single number. It is defined as

$$c_{score}(D) = \frac{\sum_{t=0}^{D} c_{value}(t)}{D}. \tag{4.2}$$

4.2.2 Throughput: Measuring Network Performance

We measure the processing time of messages on a node and aggregate it by the granularity of one second. This constitutes the throughput of a particular node at a given time. To evaluate the network's mean throughput we sum up the throughput of all nodes divided

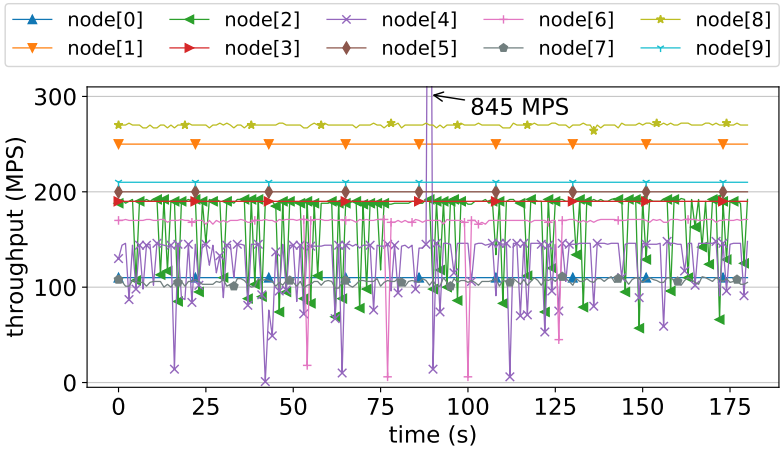

Figure 4.1: Processing rates ν_m in a network with 10 nodes. A node's mean processing rate $\nu_m \geq \nu_{net}$.

by the count of nodes. This calculation excludes out-of-sync nodes, i.e., only participating nodes contribute to the network's throughput.

4.2.3 Latency: Assessing Delays

Delays are important in every network where small delays correspond to quick response times. In voting-based DLTs a small latency is crucial for nodes to be able to participate in consensus. We measure the latency of any given node m as the duration between message issuance time and processing time on node m. When assessing latency as global network property, we consider the 95 percentile latency of all in-sync nodes.

4.3 Microbenchmarks

We compare aided and Healthor in a network with 10 nodes with available processing rates. Figure 4.1 depicts that each node's mean processing rate ν_m is larger than or equal to ν_{net}. It can be seen that `node[2]` ($\bar{\nu}_2 = 170$ MPS), `node[4]` ($\bar{\nu}_4 = 130$ MPS), and `node[6]` ($\bar{\nu}_6 = 170$ MPS) sometimes fall below the network processing rate $\nu_{net} = 100$ MPS. Therefore, these nodes are of special interest. Due to the limiting nature of aided the actual processing rates used are $\min(\nu_{net}, \nu_m)$ and higher rates can not be leveraged.

4.3.1 Comparison of Aided and Healthor

Figure 4.2 details the throughput of aided (left) and Healthor (right). The x-axis shows the throughput in messages per second, and the y-axis the time in seconds. Each node's throughput can be directly related to its available processing rate as depicted in Figure 4.1.

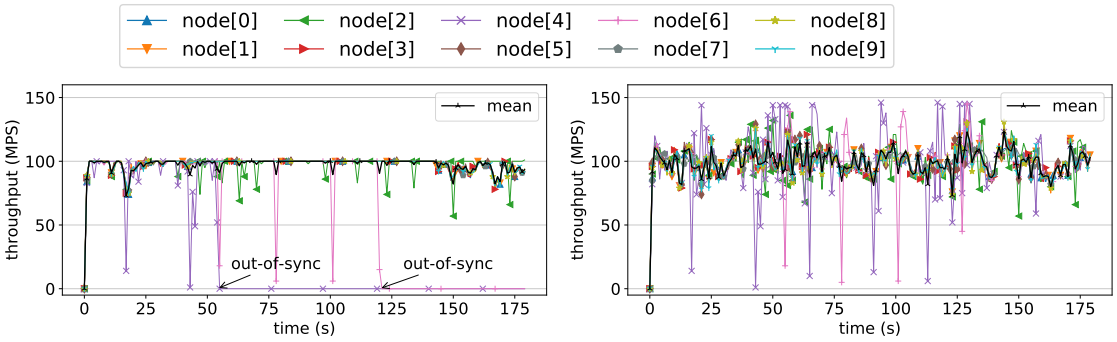

Figure 4.2: Throughput.

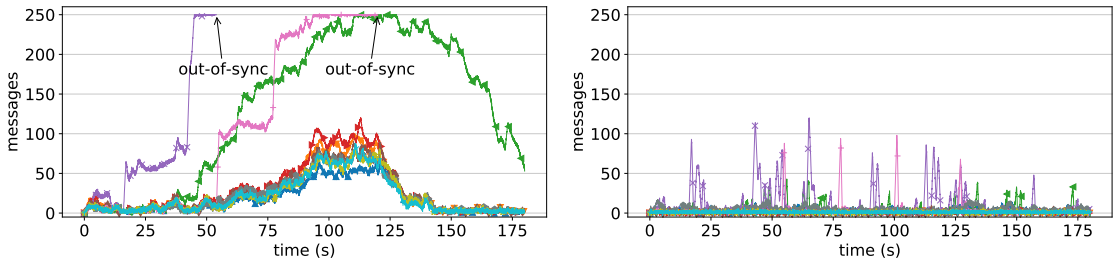

Figure 4.3: Inbox occupation.

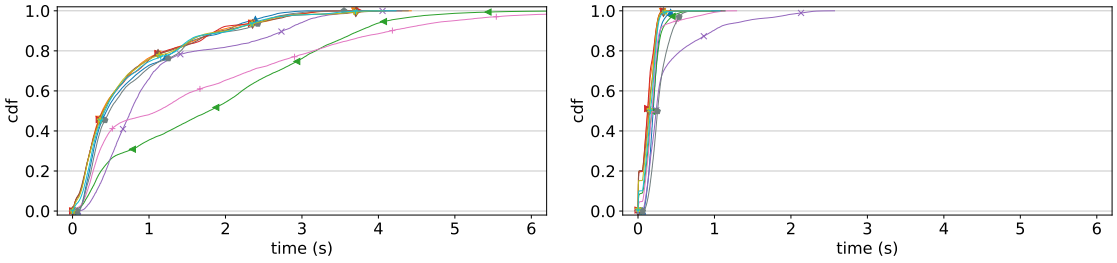

Figure 4.4: Latencies.

Figure 4.5: Experiment results in a network with 10 nodes with aided heterogeneity (left) and Healthor (right).

For aided, the throughput is capped at a maximum rate ν_{net} ($mean = 97.51$ MPS), while Healthor temporarily allows higher throughput ($mean = 99.86$ MPS) which fluctuates around—instead of being limited by—the value ν_{net}. In aided, `node[4]`'s throughput drops to 0 around second 55. Similarly, `node[6]`'s throughput falls to 0 at 120 seconds. This indicates that both nodes are out of sync, i.e., their inboxes are filled up with too many messages without their known history, so that no newly received message can be scheduled. `node[4]` and `node[6]` go offline from this point in time and cannot participate in consensus anymore until a heavy re-sync operation is performed. In practice this could mean manual intervention and restart with a trusted, previously downloaded ledger state snapshot.

Figure 4.4 shows the CDFs of the latencies for aided and Healthor, respectively. In aided, `node[2]` and `node[6]` have by far the largest latency, exceeding 5 seconds at the tail. Recalling the definition of centralization score in Section 4.2, this indicates that these nodes are too far behind to participate in consensus for some messages. Indeed, as shown in Figure 4.2, `node[4]` and `node[6]` run out of sync. However, `node[4]`'s latency does not exceed 5 seconds which implies that it got out of sync quickly. With Healthor, the network latency is significantly lower by 73% compared to aided. Especially `node[2]`'s and `node[6]`'s 95 percentile latencies stand out with an improvement of 91% and 89%, respectively.

4.3.2 A Closer Look at Inbox Sizes

Figure 4.3 shows the number of messages stored at each node's inbox over time, i.e., inbox occupation. Every node can store up to 250 messages. Generally, we observe that buffers have higher occupation in aided, which leads to higher latencies and indicates higher memory load. Messages reside longer in the inbox before they can be scheduled because of the enforced processing rate limit. `node[4]`'s and `node[6]`'s inboxes run full around second 40 and 90, respectively. Once this happens for too long the node is non-recoverable out of sync and, simultaneously, its throughput drops to 0, hence it goes offline. Temporarily, also `node[2]`'s inbox runs full at $\sim$ 120 seconds, but it does not go out of sync. Instead, it is able to recover via pull action and its inbox occupation drops towards the end of the simulation. Inbox occupation with Healthor follows a different pattern: none of the nodes' inbox runs full, thus all nodes stay in sync. Furthermore, the inbox fills and empties in a zigzag pattern reacting to health changes. Recall that a node gets unhealthier when its inbox grows, and its neighbors forward at slower rate until the node gets healthier again.

Inbox	c_{score}		MI `node[4]`		MI `node[6]`	
	aided	Healthor	aided	Healthor	aided	Healthor
100	0.14	0	100	100	100	90
250	0.11	0	250	120	250	98
500	0.08	0	500	120	400	115
1,000	0.08	0	920	120	400	115
2,000	0.08	0	920	120	400	110

Table 4.1: Different inbox sizes with aided and Healthor. MI=maximum measured inbox occupation. $c_{score} = 0$ means no node is out of sync (desired).

Table 4.1 shows the centralization score and the maximum inbox occupation of `node[4]` and `node[6]` for various inbox sizes. The centralization score captures when a node is not able to participate in consensus. Therefore, it is a reasonable tool to assess the effect of various inbox sizes. On the one hand, a too small inbox can be easily fulfilled. Hence, new messages are dropped with high probability up to the point where the node falls out of sync. On the other hand, a too big inbox size might consume too much memory while only increasing delays. Either way, a node is not able to vote.

We observe that `node[4]` and `node[6]` get out of sync in the aided case when the inbox size is 100 because both nodes' inboxes run full. With Healthor, the centralization score is 0 which signals that no nodes got out of sync, even though `node[4]`'s inbox reached 100. In this case, `node[4]` could recover via a pull action and with the help of its neighbor's buffering while it is unhealthy. Larger inbox sizes seem to improve the centralization score slightly for aided but the fundamental problem of `node[4]` and `node[6]` getting out of sync remains. With Healthor this problem is already alleviated with an inbox size of 100, where every node can stay in sync. However, `node[4]`'s inbox is at its maximum capacity, and the node needed to recover via pull actions. Inbox sizes larger than 250 only let nodes look healthier, but cannot improve the centralization score (which is already 0). Therefore, the maximum inbox occupation measured is slightly higher. Henceforth, we adopt an inbox size of 250 for all of our experiments as a reasonable tradeoff between memory consumption and storing capacity.

4.3.3 A Closer Look at Outbox Sizes with Healthor

Recall that a node m has an $Outbox_n$ for every $n \in \mathcal{N}_m$. It is therefore important to establish a fitting outbox size so that enough messages can be stored, but no unnecessary overhead is created. Table 4.2 shows the centralization score, maximum inbox occupation

Outbox	c_{score}	MI `node[4]`	MON `node[4]`	MI `node[6]`	MON `node[6]`
100	0	160	100	98	100
250	0	120	200	98	100
500	0	120	200	98	100
1,000	0	120	200	98	100
2,000	0	120	200	98	100

Table 4.2: Outbox sizes with Healthor. MI=maximum measured inbox occupation, MON=maximum outbox occupation at neighbor node. $c_{score} = 0$ means no node is out of sync (desired).

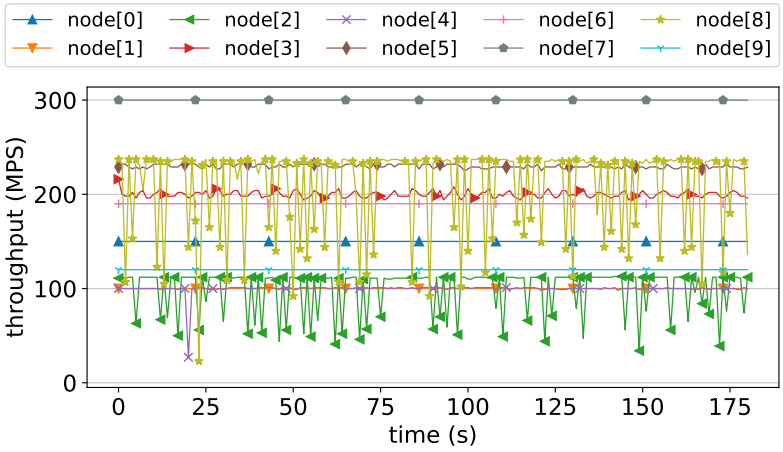

Figure 4.6: Processing rates ν_m in a network with 10 nodes (additional scenario). A node's mean processing rate $\nu_m \geq \nu_{net}$.

of `node[4]` and `node[6]` as well as maximum outbox occupation of both nodes' neighbors. Larger outboxes indicate less pressure on an unhealthy node up to an outbox size of 250, as is evident by the higher inbox occupation of `node[4]` with an outbox size of 100. However, this trend can only be observed until an outbox size of 250, where the outbox occupation at `node[4]`'s neighbors and its inbox occupation stabilize. An outbox size of 250 seems to offer a good tradeoff between decreasing load on an unhealthy node and creating overhead on its neighbors. We, therefore, adopt an outbox size of 250 for our experiments.

4.3.4 Additional Scenario

The previous analysis was done with the same network and node heterogeneity setup. To gain a better intuition of Healthor's behavior for different networks, we now take a look at the same set of figures (Figure 4.10) for throughput, inbox occupation, and latencies for a different 10 node network setup. Figure 4.6 shows the nodes' processing rates ν_m, where each node's mean processing rate ν_m is larger than or equal to ν_{net}. The network's

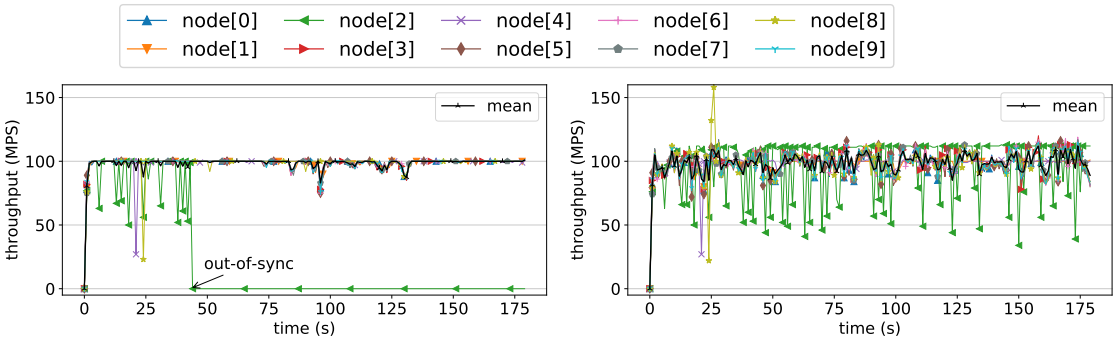

Figure 4.7: Throughput.

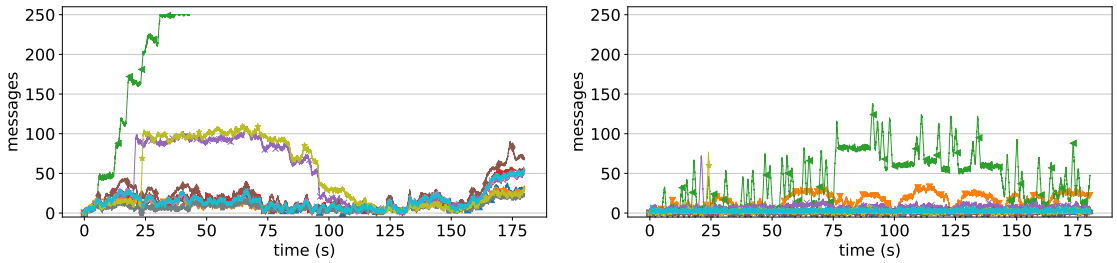

Figure 4.8: Inbox occupation.

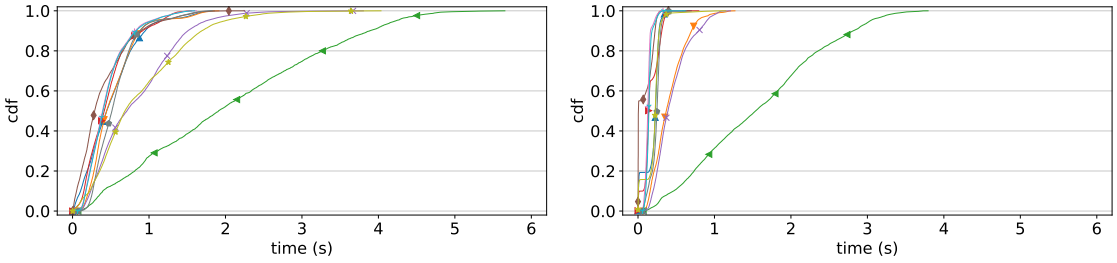

Figure 4.9: Latencies.

Figure 4.10: Experiment results in a network with 10 nodes (additional scenario) with aided heterogeneity (left) and Healthor (right).

parameter are as described in Section 4.1. We observe that the mean of v_m for `node[1]`, `node[2]`, and `node[4]` are equal to the network processing rate $\nu_{net} = 100$ MPS. Hence, these nodes are of special interest.

Figure 4.7 details the throughput of aided (left) and Healthor (right). We show, for the former, the throughput is capped at a maximum rate v_{net}, while the latter, temporarily allows higher throughput, which fluctuates around — instead of being limited by — the value v_{net}. In aided, `node[2]`'s throughput drops to 0 around second 45 which means that the node is out of sync and offline from this point in time and cannot participate in consensus anymore.

In Figure 4.8 the inbox occupation is displayed. At the same time, we observed `node[2]`'s throughput dropping to 0, i.e., getting out of sync, we can see its inbox becoming full for the aided case (left). Comparing to Healthor (right), `node[2]`'s inbox does not grow larger than 145 and follows the zigzag pattern reacting to health changes, as we have seen for the first scenario already. Overall, inboxes do not grow as full with Healthor as in aided.

The CDFs of the latencies for aided (left) and Healthor (right) are shown in Figure 4.9. In aided, `node[2]`'s latency exceeds the 5s mark and indicates, once again, that this node is getting out of sync. With Healthor the latency for `node[2]` is lower and this is not the case. For all other nodes especially tail latencies are large in aided whereas this is not the case with Healthor.

4.3.5 Overheads

In our evaluation, we configure Healthor to exchange health messages every 1 second. Naturally, this adds message overheads compared to aided. The theoretical maximum overhead of a node for a simulation of length T time can be calculated as $o_{max}(m) = |\mathcal{N}_m|T$. Assuming that a node has $|\mathcal{N}_m| = 4$ neighbors during a simulation of 180 seconds results in $o_{max} = 720$ health messages sent and received. In a flooding-based P2P network with $\nu_{net} = 100$ MPS and the same configuration the maximum number of sent and received messages is 72,000. It follows that Healthor incurs a maximum overhead of 1% in the deployed configuration. However, we are aware that the overhead is a function of message rate and can be high with low throughput.

4.3.6 Discussion

In two different scenarios, we have shown that Healthor enables lower latencies and reduces load on low-end nodes compared to aided mainly by allowing temporarily higher throughput. It, therefore, unlocks enormous potential: *nodes can make use of resources*

$\|\mathcal{M}\|$	aided			unaided			Healthor		
	c_{score}	**L**	**T**	c_{score}	**L**	**T**	c_{score}	**L**	**T**
100	0.04	1.93	98.24	0.40	0.83	198.38	0.02	0.73	101.10
500	0.09	4.24	97.15	0.38	1.17	196.20	0.02	0.95	100.81
1,000	0.07	4.47	97.82	0.38	1.45	195.20	0.03	0.99	100.12
2,000	0.06	3.17	97.79	0.41	1.61	196.98	0.03	1.06	100.82
5,000	0.07	4.26	97.00	0.40	1.73	197.21	0.03	1.19	100.60

Table 4.3: Decentralization vs. throughput experiment results. L=95 percentile latency, T=mean throughput.

when they are available, irrespective of network activity. However, the mechanism does not come without overheads as our calculations show.

4.4 Macrobenchmarks

In the previous section, we investigated Healthor closely and established sensible default parameters for inbox and outbox sizes. Following, we compare aided, unaided, and Healthor at a larger scale. We present overall network-level results for heterogeneous networks with up to 5,000 nodes in line with the description provided in Section 4.1.

4.4.1 Decentralization vs. Throughput

Figure 4.11 shows the centralization value for aided, unaided, and Healthor in a network with 2,000 nodes. Clearly, unaided is the least decentralized whereas aided is significantly more decentralized and Healthor even more so. Figure 4.12 shows the throughput. As expected, aided is limited at $\nu_{net} = 100$, Healthor permits temporarily higher throughput around ν_{net}, and unaided is only limited by demand (here at $\nu_{net} = 200$). In Figure 4.13, the CDF of 95th percentile delay of in-sync nodes details how delays are much higher in aided than in unaided ($\sim 2x$ higher) and Healthor ($\sim 3x$ higher). Messages in aided stay much longer in a buffer before being able to be processed and forwarded due to the limited rate. It is interesting to see that while Healthor slightly increases throughput and makes the network significantly more decentralized compared to aided, it still provides comparable latencies to unaided.

A similar trend can be observed in Table 4.3. In various network settings, unaided allows the highest throughput and has low latency, but it is also the least decentralized. Aided guarantees a fair decentralization but has poor latency and a maximum fixed throughput.

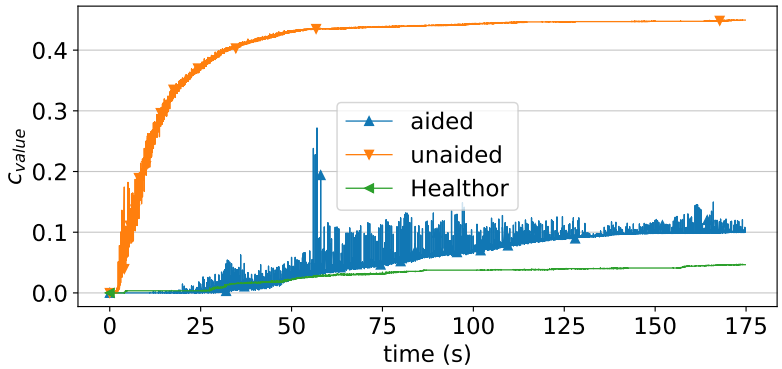

Figure 4.11: Centralization value.

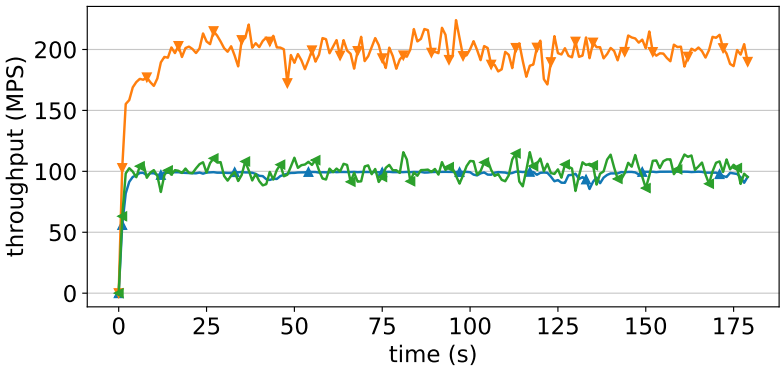

Figure 4.12: Throughput.

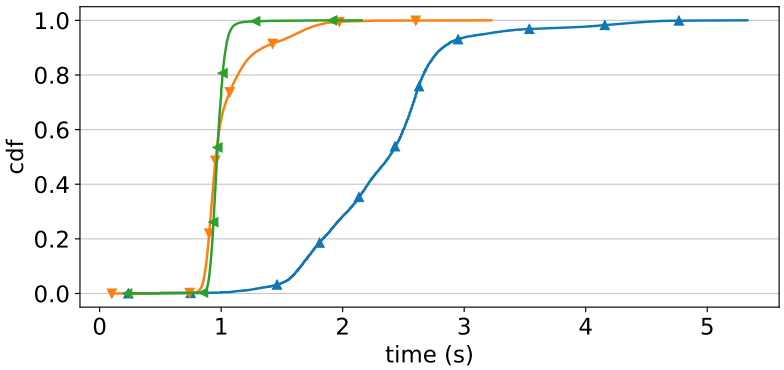

Figure 4.13: 95 percentile latency.

Figure 4.14: Experiment results in a network with 2,000 nodes comparing aided, unaided and Healthor. The graphs show differences in performance and decentralization tradeoff where Healthor provides a reasonable middleground.

$\|\mathcal{M}\|$	c_{score}	Latency			Throughput		
		aided	**Healthor**	**Dec. (↓)**	**aided**	**Healthor**	**Inc. (↑)**
100	0.04	1.93	0.76	61%	98.24	105.45	7%
500	0.09	4.24	1.06	75%	97.15	119.51	23%
1,000	0.07	4.47	1.08	76%	97.82	115.10	18%
2,000	0.06	3.17	1.12	65%	97.79	108.96	11%

Table 4.4: Comparing throughput and latency in aided and Healthor with same degree of decentralization.

Healthor offers the best out of both worlds: almost complete decentralization (78% improvement compared to aided), a slightly better throughput than aided and its latency is on par with unaided.

4.4.2 Sensitivity Analysis

Looking closer at the behavior of aided and Healthor with the same degree of decentralization can reveal in which settings our mechanism improves throughput and latency. This is shown in Figure 4.15, where throughput (left) and latency (right) are plotted on the y-axis and centralization score on the x-axis. As expected, the throughput of aided converges towards ν_{net} and cannot exceed it with increasing centralization scores. We basically observe the system breaking down because there are more messages issued than the fixed throughput ν_{net}. With Healthor it looks decidedly different: the throughput correlates almost linearly with the centralization score, hence is not limited at a fixed rate (desired). The network remains functional, however, it is in a similar elitist mode as unaided, where only high-end nodes can continue to participate. In Figure 4.15 (right) we observe a correlation between rising latency and increasing centralization score in aided (undesired). Conversely, with Healthor we see a flat latency, hovering around 1 second, regardless of increasing centralization scores.

Table 4.4 shows by how much Healthor improves latency and throughput compared to aided with the same degree of decentralization for networks of various sizes. Again, we can observe much higher latencies with aided and a significant improvement of up to 76% with Healthor. Similarly, Healthor improves the throughput by up to 23% compared to aided in a network with 500 nodes.

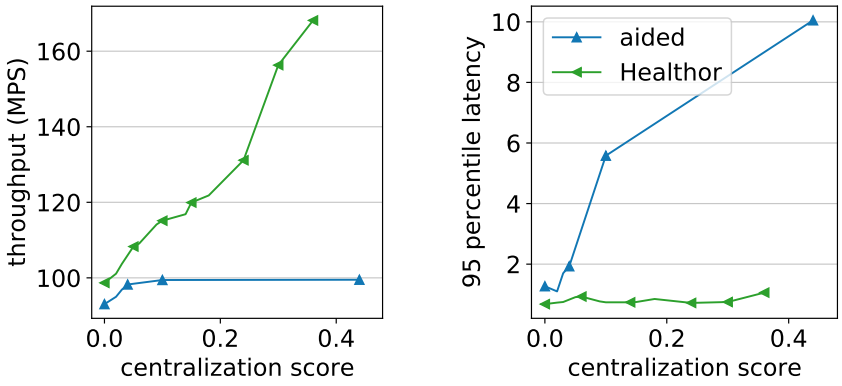

Figure 4.15: Sensitivity analysis of unaided compared to Healthor in a network with 100 nodes. Left: Correlation throughput and c_{score} (higher is better). Right: Correlation latency and c_{score} (lower is better).

4.4.3 Discussion

Aided shows one side of the extreme and limits throughput to increase decentralization. In light of processing heterogeneity, however, this is not very effective as our experiments have shown. The problem is that nodes may experience temporary slowdowns below the network processing rate ν_{net}. As long as the throughput is close to ν_{net}, these nodes lag behind further and further as they experience slowdowns creating higher latency. On the other hand, the limit on throughput prevents these nodes from falling out of sync completely and therefore provides decentralization.

With unaided nodes can simply operate at their own speed at all times. This clearly offers lower latency, but it creates an elite-network where only the strongest high-end nodes can participate, thus reduces decentralization. Since throughput is only limited by demand, it is practically only limited by the processing capabilities of the fastest participant(s) at the cost of excluding slower devices and becoming more centralized.

Healthor provides the best out of both worlds by embracing heterogeneity. It allows nodes to temporarily go faster than ν_{net} and hence offers low latencies comparable to unaided, higher throughput than aided, and most importantly improves decentralization even more than aided.

4.5 Attack Analysis

Permissionless P2P systems are required to face challenges posed by adversarial behavior. In the case of DLTs, this is exacerbated through financial motives. Therefore, protocols in permissionless DLTs should be tamperproof and resilient against exploitation. Healthor employs several inherent defense mechanisms as described in Section 3.7. We evaluate how the *Defense Engine* operates in targeted attacks.

4.5.1 Attack Model

In the presented attacks we assume an omniscient adversary. Hence, the adversary is aware of every node's health, can intercept both messages and health messages. She is able to tamper with the aspects of Healthor's protocol. Attacks on the network topology, such as eclipse attacks, and the underlying ledger are out of scope of this thesis. We use the same 10 nodes network and parameters of Section 4.3 for the attacks shown in this section.

4.5.2 Manipulation of Health Updates

Not sending health updates. Recall that a node assumes a neighbor n to be healthy $h_n = 1$ when connecting to each other. Further, it uses the last known health status if a neighbor does not report any health. Therefore, an adversary cannot cause any harm by withholding health updates.

Lying about health. Since health is computed locally and then sent to the neighbors a node can lie about its health and even send different health status to distinct neighbors. From the viewpoint of a neighbor n of such an adversarial node m_a, node m_a behaves normally as long as no other protocol violations are detected. In essence, an adversary has only very limited capabilities which only affect itself.

Pretending to be unhealthy. An adversary could try to inflate outboxes on its neighbors by pretending to be unhealthy. Healthor intrinsically averts this by employing limited lengths for buffers. However, an adversary could make its neighbors waste resources up to this limited amount which is not favorable. The *Defense Engine* implements a mechanism to detect whether a node is unhealthy for too long indirectly via its outbox occupation. If the outbox of a neighbor n is too full for too long, it is dropped. Figure 4.16 shows this scenario where `node[1]` attacks `node[3]`. $Outbox_{node[1]}$ grows rapidly up to the maximum capacity of 250. Eventually, `node[3]` drops `node[1]` because the outbox has been full for too long. It can now repeer with another node.

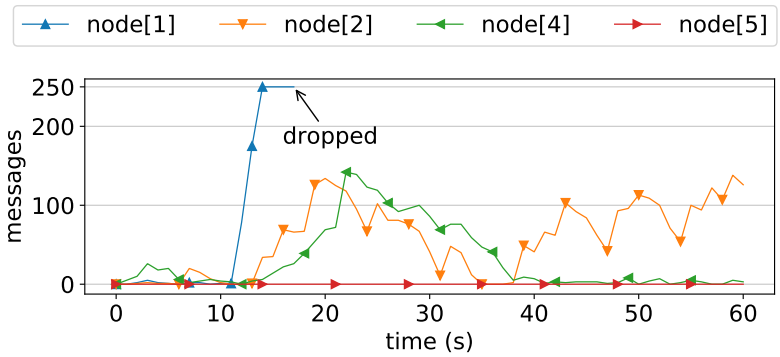

Figure 4.16: `Node[3]`'s outbox occupation. `Node[1]` attacks `node[3]` by pretending to be unhealthy.

4.5.3 Manipulation of Forwarding Rate

Forwarding more than allowed rate to neighbor. An adversary can choose to diverge from the allocated forwarding rate by sending more and trying to overload a victim. This is displayed in Figure 4.17 where `node[7]` attacks `node[2]` without *Defense Engine*. The figure shows the forwarding rates of `node[2]`'s neighbors as recorded on receipt by `node[2]`. The blue line on top indicates the maximum allowed forwarding rate for any neighbor. It is clear that `node[7]` exceeds this rate starting around second 15. As `node[7]` continues to send at high rate, `node[2]` gets unhealthier. Thus, the maximum allowed forwarding rate drops, and other neighbors of `node[2]` reduce their rate accordingly. As a result, we can see that without the *Defense Engine* an attacker can overload a node by forwarding above the allowed rate.

Figure 4.18 shows the same attack but with the *Defense Engine* enabled. `Node[2]` quickly detects `node[7]` divergence and drops the connection to it around second 20. This results in `node[2]` staying healthy and being able to process messages by other nodes until the end of the simulation.

Forwarding too little to neighbor. A node might receive none or very few messages from a specific neighbor. This could be due to a neighbor very unhealthy or a targeted attack to slow down and exclude the node from participating. Exactly this attack is pictured in Figure 4.19. At second 40 `node[7]` slows down its forwarding rate to `node[2]` to 10 MPS which is shortly after detected by `node[2]`. The connection is dropped and `node[2]` can peer with another node.

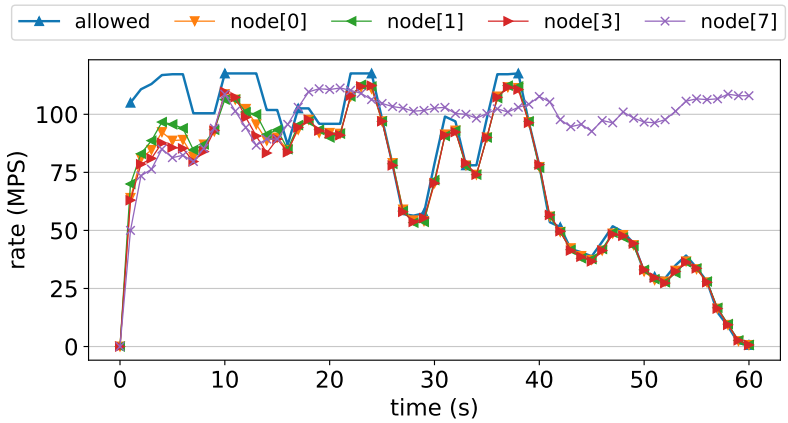

Figure 4.17: Disabled *Defense Engine*. `Node[7]` exceeds allowed forwarding rate.

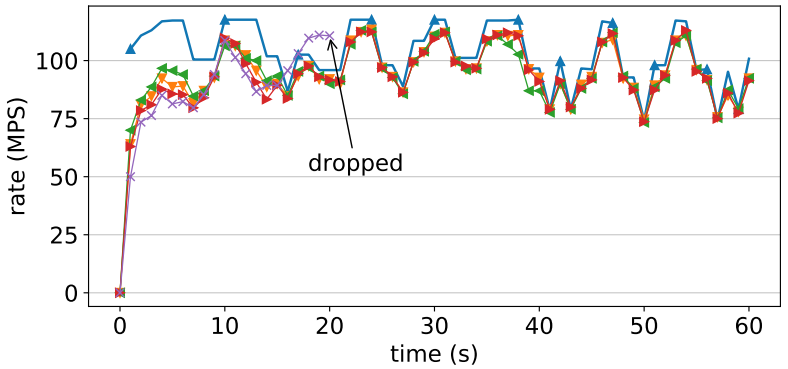

Figure 4.18: Enabled *Defense Engine*. Exceeding of forwarding rate is quickly detected.

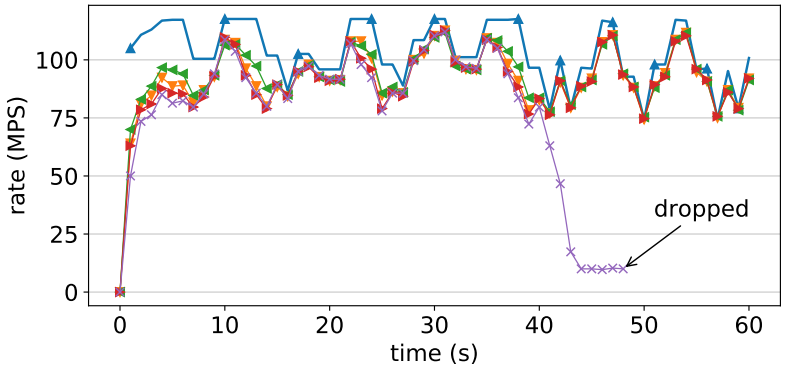

Figure 4.19: Enabled *Defense Engine*. Slowing down substantially is quickly detected.

Figure 4.20: Receiving rates of `node[2]`'s neighbors, measured locally. `Node[7]` attacks `node[2]`.

4.6 Summary

In this chapter we introduced a new metric called the *centralization score* to quantify the degree of centralization in DLTs (Section 4.2). We evaluated Healthor at micro (Section 4.3) and macro (Section 4.4) scale, explored node and network properties and tested its behavior when faced by adversaries (Section 4.5).

Microbenchmarks show that Healthor takes load away from low-end nodes and allows these nodes to stay in sync longer compared with aided heterogeneity. In our macrobenchmarks, we find that Healthor is able to increase the degree of decentralization by 78%, improve throughput by 23%, and decrease 95 percentile message latency by 4×. Finally, we observe Healthor's resiliency against malicious behavior in the in-depth attack analysis. Overall, Healthor performs very well but adds message overhead as a function of the message rate, i.e., a high network rate means low relative overhead whereas a low network rate means higher relative overheads.

Chapter 5

Related Work

Flow control in the permissionless DLT setting builds on many topics from networking research such as P2P systems, network security, and distributed flow control. However, to the best of our knowledge Healthor is the first system to systematically combine a health-based, heterogeneity-aware distributed flow-control protocol in permissionless DLTs.

5.1 Traditional Networks

Congestion and flow control in traditional networks and data centers is a deeply studied topic (30, 46, 47, 48, 49, 98). In traditional networks, mostly end-to-end communication settings rather than group communication is considered. However, with Healthor we focus on message flow control in a group communication setting. Due to the open nature of permissionless DLTs, it is not feasible to rely on hardware or operating system support. Therefore, Healthor operates on the application level.

Generally, congestion control protocols employ specific signals such as delay measurements or packet loss to detect congestion. Especially protocols designed for data centers often require hardware support, e.g., switch support in DCTCP (46) and NIC support in TIMELY (30). Other protocols such as TCP Cubic (47), Copa (48), and BBR (49) are entirely software-based, often provide kernel modules or are shipped and integrated into popular operating systems.

In Healthor we use explicit health messages from neighbors as a flow-control signal without needing any hardware assistance. The Healthor architecture converts these messages to a forwarding rate per neighbor, thus Healthor is a rate-based mechanism. The engine framework is inspired by TIMELY (30), which itself is a rate-based congestion control mechanism (in contrast to window-based TCP variants (46, 47)).

5.2 P2P Systems

P2P systems and specifically content distribution systems such as Gnutella, KaZaA and BitTorrent were studied widely in the early 2000s (27, 28, 29, 55, 56, 57). Similar to modern DLTs, these systems are highly heterogeneous (22) and faced challenges with scalability and performance (27). In (26, 59) Gia, a scaling approach for the Gnutella network, is proposed that leverages heterogeneity. It dynamically changes the network topology and puts high capacity nodes within short reach of most nodes. Additionally, an active flow-control mechanism based on available capacity is used to avoid overloaded hot-spots. Gia is similar to Healthor in spirit as both mechanisms leverage heterogeneity to increase utilization. However, the way Gia is designed is unsuitable for permissionless DLTs as topology changes can lead to cliques construction of powerful nodes, thus defeating the aim of decentralization in DLTs. Also, in Gnutella, it is not essential for nodes to retrieve every search query whereas in DLTs state updates need to be propagated to every node eventually.

Next to challenges around scalability and performance, content distribution systems have been shown to suffer from free-riding, i.e., the lack of cooperation between peers (22, 29, 55, 64). If no proper incentives are provided, users tend to misreport information, act selfishly and even exploit other users (22). Therefore, many P2P systems like BitTorrent (29) incorporate game-theoretic incentives, e.g., based on tit-for-tat (65), or reputation systems (68, 69) to preclude such malicious behavior. In Healthor, we provide incentives to follow the protocol through the Defense Engine. Deviations from allowed forwarding rates lead to exclusion of the network eventually, because neighbors will drop peers that do not adhere to the allowed rates. In future work, behavior on the network level within Healthor's protocol could be integrated into the underlying DLT's reputation and thus provide much more incentives to be honest and penalize malicious actions.

5.3 DLT

More recent research on DLTs focuses more on the aspects and security of consensus (9), decentralization (3, 12, 77), and scalability (87). Scalability research in DLTs is mainly concerned with scaling consensus, often called sharding, where not all network participants need to process all ledger state updates to save resources and lower cost. Only little research has been done concerning the network layer (86). There are some proposals to improve transaction and block dissemination such as Kadcast (99) and Erlay (100).

Heterogeneity, however, is often not considered at all in many DLTs' protocols but, e.g., in PoW DLTs where mining races exist, often not beneficial for users (9). Studies that investigate the decentralization of the underlying P2P network of DLTs do not take heterogeneity into account in terms of who is capable of participating, instead it is merely considered a by-product caused by performance variability of cloud providers (2, 23, 24). With Healthor, we provide a solution to this emerging problem and show that this heterogeneity can be harmful for decentralization if not taken into account. However, it can be leveraged as we show with Healthor.

As an alternative to public permissionless DLTs, there are also permissioned DLT designs (101) that can potentially give more control over participants and their network activities.

Chapter 6

Conclusion

In this thesis, we have presented Healthor, one of the first distributed flow-control mechanisms to leverage node heterogeneity to dynamically improve performance and decentralization in a permissionless DLT network. Healthor achieves this by rate-controlling message forwarding rates based on a node's health. Health is defined as a function of a node's message inbox occupancy. With extensive simulation of Healthor on DLT networks of up to 5,000 nodes, we have shown that this simple health-based signal can increase the degree of decentralization by 78%, improves throughput by 23%, and decreases 95 percentile message latency by 4×. Healthor's source code (`https://github.com/jonastheis/healthor`) and simulation result data set (`https://zenodo.org/record/4573698`) are both publicly available.

6.1 Answering Research Questions

1. How to derive a right design for a dynamic, distributed flow-control protocol?

In Chapter 2 and Chapter 5 we reviewed fundamentals and related work, respectively. Inspired by TIMELY's (30) rate-control framework with separate engines and Gia (26) we designed Healthor. In iterative steps the engines evolved into their final form while being prototyped in the simulator. In small-scale simulations the behavior was inspected, reiterated, and the design adjusted accordingly.

2. What are the key challenges in designing a dynamic, distributed flow-control protocol in the permissionless DLT setting?

Distributed protocols pose hard challenges due to limited knowledge of individual nodes, especially in the permissionless DLT setting where adversaries have lucrative monetary

incentives to attack the protocol. It is therefore important to design a resilient and tamperproof protocol. In the case of a dynamic, distributed flow-control mechanism the *signal* to rate control is paramount. On the one hand, it needs to be resistant to cheating and capture heterogeneity as well as ever-changing processing capabilities of nodes. On the other hand, it should provide fair rates and utilize available resources fully. Identifying these parameters, measuring them, and finally finding the right balance is the key challenge when designing a dynamic, distributed flow-control protocol in the permissionless DLT setting.

In Healthor, *health* covers most of these aspects as we describe in Chapter 3 and evaluate in Chapter 4. However, the notion of *health* and its translation to a forwarding rate does not guarantee full utilization as some nodes might not be able to send at their allocated, fair rate.

3. How efficient is such a dynamic flow-control protocol for permissionless reputation-based DLTs?

In Chapter 4 we compare Healthor against the baseline scenario with aided heterogeneity in networks of up to 5,000 nodes. Healthor is able to increase the degree of decentralization by 78%, improve throughput by 23%, and decrease 95 percentile message latency by 4× while staying resilient against attacks. Healthor adds message overhead as a function of the message rate, i.e., a high network rate means low relative overhead whereas a low network rate means higher relative overheads.

6.2 Limitations and Future Work

6.2.1 Operating Without Network Processing Rate

The fewer parameters need to be specified in a protocol a priori, the more efficient and resilient the protocol becomes. This also applies to the network processing rate ν_{net}. The capabilities of participating nodes can change over time and an optimal solution would not assume a target rate. Instead, a self-stabilizing mechanism where nodes dynamically smooth network activity to an equilibrium based on network health and network participants would be more powerful. Such a solution allows network components to be upgraded over time without the daunting exercise to change the protocol. A protocol update inherently bears the risk of a network split if participants do not upgrade or refuse to upgrade, and can therefore be problematic in the permissionless setting.

6.2.2 Right Incentives

Like early P2P designs, targeted incentives are needed to promote *protocol-following good behavior* in DLTs. However, unlike classical DLTs such as Bitcoin, feeless reputation-based DLTs by-design do not offer any monetary incentives such as mining or minting fees. Similarly, Healthor does not provide hardened incentives for nodes to follow the protocol except risking being dropped by a neighbor when not adhering to the protocol. One potential future design is to incorporate Healthor's neighbor-assisting behavior into a DLT's reputation system incentives. In that way, nodes can be rewarded for good behavior and penalized for bad behavior in terms of their score for consensus.

6.2.3 Variable Forwarding Rate

The Rate Computation Engine as described in Section 3.4 calculates the same rate r_n on every neighbor of node n. While this provides an easy way of distributing the rate equally, it might lead to underutilization if a neighbor of n is not able to send at its fair share. Therefore, a dynamic rate-allocation mechanism between neighbors is an interesting avenue to explore. This, however, is not trivial since such a mechanism needs to work in a distributed setting without opening new attack vectors.

6.2.4 Real-world Deployment

In this thesis experiments and results are produced using a simulator written in OMNeT++. While many aspects like peering and a distributed ledger are modeled, others are not covered by the simulation environment. For example, the actual computational overhead created by Healthor is not measurable in the simulator. Many of the operations seem to be lightweight but time-based measurements in a real-world setting could clarify concerns while utilizing concurrency.

6.2.5 Other Use Cases

Healthor can not only be deployed in a DLT setting. Other use cases where devices have limited capabilities such as Internet of things and edge computing could benefit from it as well. However, currently Healthor relies on a chain-like data structure to avoid consistency issues when messages are dropped.

References

[1] Maurice Herlihy. **Blockchains and the Future of Distributed Computing**. In *Proceedings of the ACM Symposium on Principles of Distributed Computing*, PODC '17, page 155, New York, NY, USA, July 2017. Association for Computing Machinery. ii

[2] Jamie Ericson, Masoud Mohammadian, and Fabiana Santana. **Analysis of Performance Variability in Public Cloud Computing**. In *2017 IEEE International Conference on Information Reuse and Integration (IRI)*, pages 308–314, San Diego, CA, August 2017. IEEE. vii, 2, 3, 18, 49

[3] Seoung Kyun Kim, Zane Ma, Siddharth Murali, Joshua Mason, Andrew Miller, and Michael Bailey. **Measuring Ethereum Network Peers**. In *Proceedings of the Internet Measurement Conference 2018*, pages 91–104, Boston MA USA, October 2018. ACM. xi, 2, 12, 14, 30, 31, 48

[4] Satoshi Nakamoto. **Bitcoin: A Peer-to-Peer Electronic Cash System**. page 9, 2009. 1, 11, 16

[5] Roger Wattenhofer. *Distributed Ledger Technology: The Science of the Blockchain*. CreateSpace Independent Publishing Platform, North Charleston, SC, USA, second edition, 2017. 1, 13

[6] Vikas Hassija, Gaurang Bansal, Vinay Chamola, Neeraj Kumar, and Mohsen Guizani. **Secure Lending: Blockchain and Prospect Theory-Based Decentralized Credit Scoring Model**. *IEEE Transactions on Network Science and Engineering*, **7**(4):2566–2575, 2020. 1

[7] Nabil El Ioini and Claus Pahl. **A Review of Distributed Ledger Technologies**. In Hervé Panetto, Christophe Debruyne, Henderik A. Proper,

Claudio Agostino Ardagna, Dumitru Roman, and Robert Meersman, editors, *On the Move to Meaningful Internet Systems. OTM 2018 Conferences*, **11230**, pages 277–288. Springer International Publishing, Cham, 2018. 1, 11, 13, 15

[8] Ethereum. **Ethereum/Devp2p**, 2020 [cited 2021-01-23]. 1, 12, 30

[9] Florian Tschorsch and Bjorn Scheuermann. **Bitcoin and Beyond: A Technical Survey on Decentralized Digital Currencies**. *IEEE Communications Surveys & Tutorials*, **18**(3):2084–2123, 23. 1, 11, 12, 13, 15, 16, 17, 31, 48, 49

[10] Serguei Popov, Hans Moog, Darcy Camargo, Angelo Capossele, Vassil Dimitrov, Alon Gal, Andrew Greve, Bartosz Kusmierz, Sebastian Mueller, Andreas Penzkofer, et al. **The Coordicide**. 2020. 1, 3, 12, 16, 17, 20, 31

[11] Arthur Gervais, Ghassan O. Karame, Karl Wüst, Vasileios Glykantzis, Hubert Ritzdorf, and Srdjan Capkun. **On the Security and Performance of Proof of Work Blockchains**. In *ACM CCS*, pages 3–16, 2016. 1

[12] Adem Efe Gencer, Soumya Basu, Ittay Eyal, Robbert van Renesse, and Emin Gün Sirer. **Decentralization in Bitcoin and Ethereum Networks**. In Sarah Meiklejohn and Kazue Sako, editors, *Financial Cryptography and Data Security*, **10957**, pages 439–457. Springer Berlin Heidelberg, Berlin, Heidelberg, 2018. 1, 12, 30, 31, 48

[13] Michael Bedford Taylor. **The Evolution of Bitcoin Hardware**. *Computer*, **50**(9):58–66, 2017. 1, 2, 16

[14] Ittay Eyal, Adem Efe Gencer, Emin Gün Sirer, and Robbert van Renesse. **Bitcoin-Ng: A Scalable Blockchain Protocol**. In *USENIX NSDI*, pages 45–59, 2016. 1, 17

[15] Yossi Gilad, Rotem Hemo, Silvio Micali, Georgios Vlachos, and Nickolai Zeldovich. **Algorand: Scaling Byzantine Agreements for Cryptocurrencies**. In *ACM SOSP*, pages 51–68, 2017. 1, 17

[16] Mahdi Zamani, Mahnush Movahedi, and Mariana Raykova. **RapidChain: Scaling Blockchain via Full Sharding**. In *ACM CCS*, pages 931–948, 2018. 1, 17

[17] Fahad Saleh. **Blockchain Without Waste: Proof-of-Stake**. SSRN Scholarly Paper ID 3183935, Social Science Research Network, Rochester, NY, July 2020. 1, 3, 16

[18] C. Huang, Z. Wang, H. Chen, Q. Hu, Q. Zhang, W. Wang, and X. Guan. **RepChain: A Reputation Based Secure, Fast and High Incentive Blockchain System via Sharding**. *IEEE Internet of Things Journal*, pages 1–1, 2020. 1

[19] Gavin Wood. **Polkadot: Vision for a Heterogeneous Multi-Chain Framework**. 2016. 1, 3, 16, 17, 31

[20] S. Popov. **The Tangle**. 2015. 1, 13, 15

[21] Federico Matteo Bencic and Ivana Podnar Zarko. **Distributed Ledger Technology: Blockchain Compared to Directed Acyclic Graph**. In *2018 IEEE 38th International Conference on Distributed Computing Systems (ICDCS)*, pages 1569–1570, Vienna, July 2018. IEEE. 1, 3, 13, 17

[22] Stefan Saroiu, P Krishna Gummadi, and Steven D Gribble. **Measurement Study of Peer-to-Peer File Sharing Systems**. In *Multimedia Computing and Networking 2002*, **4673**, pages 156–170, 2001. 2, 3, 10, 11, 48

[23] Yunjing Xu, Zachary Musgrave, Brian Noble, and Michael Bailey. **Bobtail: Avoiding Long Tails in the Cloud**. In *10th USENIX Symposium on Networked Systems Design and Implementation (NSDI 13)*, pages 329–341, Lombard, IL, April 2013. USENIX Association. 2, 3, 18, 49

[24] Ang Li, Xiaowei Yang, Srikanth Kandula, and Ming Zhang. **CloudCmp: Comparing Public Cloud Providers**. In *Proceedings of the 10th Annual Conference on Internet Measurement - IMC '10*, page 1, Melbourne, Australia, 2010. ACM Press. 2, 3, 18, 49

[25] Matthias Fitzi, Peter Gazi, Aggelos Kiayias, and Alexander Russell. **Proof-of-Stake Blockchain Protocols with near-Optimal Throughput**. *IACR Cryptol. ePrint Arch.*, **2020**:37, 2020. 3, 16

[26] Yatin Chawathe, Sylvia Ratnasamy, Lee Breslau, Nick Lanham, and Scott Shenker. **Making Gnutella-like P2P Systems Scalable**. In *Proceedings*

of the 2003 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications, SIGCOMM '03, pages 407–418, New York, NY, USA, 2003. Association for Computing Machinery. 3, 10, 11, 48, 51

[27] Stephanos Androutsellis-Theotokis and Diomidis Spinellis. **A Survey of Peer-to-Peer Content Distribution Technologies**. *ACM Computing Surveys*, **36**(4):335–371, December 2004. 3, 8, 9, 10, 48

[28] N. Leibowitz, M. Ripeanu, and A. Wierzbicki. **Deconstructing the Kazaa Network**. In *Proceedings the Third IEEE Workshop on Internet Applications. WIAPP 2003*, pages 112–120, San Jose, CA, USA, 2003. IEEE Comput. Soc. 3, 8, 10, 11, 48

[29] Bram Cohen. **Incentives Build Robustness in BitTorrent**. In *Workshop on Economics of Peer-to-Peer Systems*, **6**, pages 68–72. Berkeley, CA, USA, 2003. 3, 10, 48

[30] Radhika Mittal, David Zats, Vinh The Lam, Nandita Dukkipati, Emily Blem, Hassan Wassel, Monia Ghobadi, Amin Vahdat, Yaogong Wang, and David Wetherall. **TIMELY: RTT-Based Congestion Control for the Datacenter**. In *Proceedings of the 2015 ACM Conference on Special Interest Group on Data Communication - SIGCOMM '15*, pages 537–550, London, United Kingdom, 2015. ACM Press. 3, 8, 23, 47, 51

[31] A. Iosup, L. Versluis, A. Trivedi, E. van Eyk, L. Toader, V. van Beek, G. Frascaria, A. Musaafir, and S. Talluri. **The AtLarge Vision on the Design of Distributed Systems and Ecosystems**. In *2019 IEEE 39th International Conference on Distributed Computing Systems (ICDCS)*, pages 1765–1776, July 2019. 4

[32] Richard R Hamming. *Art of Doing Science and Engineering: Learning to Learn*. CRC Press, 1997. 4

[33] Ken Peffers, Tuure Tuunanen, Marcus Rothenberger, and Samir Chatterjee. **A Design Science Research Methodology for Information Systems Research**. *Journal of Management Information Systems*, **24**(3):45–77, December 2007. 4

[34] Raj Jain. *The Art of Computer Systems Performance Analysis*. John Wiley & Sons, 2008. 4

[35] Gernot Heiser. **Systems Benchmarking Crimes**. 2018. 4

[36] John Ousterhout. **Always Measure One Level Deeper**. *Communications of the ACM*, **61**(7):74–83, June 2018. 4

[37] Sonja Bezjak, April Clyburne-Sherin, Philipp Conzett, Pedro Fernandes, Edit Görögh, Kerstin Helbig, Bianca Kramer, Ignasi Labastida, Kyle Niemeyer, Fotis Psomopoulos, Tony Ross-Hellauer, René Schneider, Jon Tennant, Ellen Verbakel, Helene Brinken, and Lambert Heller. *Open Science Training Handbook*. Zenodo, April 2018. 4

[38] Mark D. Wilkinson, Michel Dumontier, IJsbrand Jan Aalbersberg, Gabrielle Appleton, Myles Axton, Arie Baak, Niklas Blomberg, Jan-Willem Boiten, Luiz Bonino da Silva Santos, Philip E. Bourne, Jildau Bouwman, Anthony J. Brookes, Tim Clark, Mercè Crosas, Ingrid Dillo, Olivier Dumon, Scott Edmunds, Chris T. Evelo, Richard Finkers, Alejandra Gonzalez-Beltran, Alasdair J. G. Gray, Paul Groth, Carole Goble, Jeffrey S. Grethe, Jaap Heringa, Peter A. C. 't Hoen, Rob Hooft, Tobias Kuhn, Ruben Kok, Joost Kok, Scott J. Lusher, Maryann E. Martone, Albert Mons, Abel L. Packer, Bengt Persson, Philippe Rocca-Serra, Marco Roos, Rene van Schaik, Susanna-Assunta Sansone, Erik Schultes, Thierry Sengstag, Ted Slater, George Strawn, Morris A. Swertz, Mark Thompson, Johan van der Lei, Erik van Mulligen, Jan Velterop, Andra Waagmeester, Peter Wittenburg, Katherine Wolstencroft, Jun Zhao, and Barend Mons. **The FAIR Guiding Principles for Scientific Data Management and Stewardship**. *Scientific Data*, **3**(1):160018, March 2016. 4

[39] Emery D. Berger, Stephen M. Blackburn, Matthias Hauswirth, Michael W. Hicks on Aug 28, and 2019. **A Checklist Manifesto for Empirical Evaluation: A Preemptive Strike Against a Replication Crisis in Computer Science**, August 2019. 4

[40] Alexandru Uta, Alexandru Custura, Dmitry Duplyakin, Ivo Jimenez, Jan Rellermeyer, Carlos Maltzahn, Robert Ricci, and Alexandru Iosup. **Is Big Data Performance Reproducible in Modern Cloud Networks?** In *17th {USENIX} Symposium on Networked Systems Design and Implementation ({NSDI} 20)*, pages 513–527, 2020. 4

[41] Jonas Theis, Luigi Vigneri, Lin Wang, and Animesh Trivedi. **Healthor: Protecting the Weak in Heterogeneous DLTs with Health-Aware Flow Control**. In *Proceedings of the 4th Workshop on Scalable and Resilient Infrastructures for Distributed Ledgers*, SERIAL'20, pages 5–8, New York, NY, USA, 2020. Association for Computing Machinery. 5

[42] cTuning Foundation. **Reproducible Papers with Artifacts** [cited 2021-03-19]. 6

[43] cTuning Foundation. **Ctuning/Ck-Artifact-Evaluation** [cited 2021-03-19]. 6

[44] J. H. Saltzer, D. P. Reed, and D. D. Clark. **End-to-End Arguments in System Design**. *ACM Transactions on Computer Systems (TOCS)*, **2**(4):277–288, November 1984. 7, 8

[45] V. Jacobson. **Congestion Avoidance and Control**. In *Symposium Proceedings on Communications Architectures and Protocols - SIGCOMM '88*, pages 314–329, Stanford, California, United States, 1988. ACM Press. 7, 8

[46] Mohammad Alizadeh, Albert Greenberg, David A. Maltz, Jitendra Padhye, Parveen Patel, Balaji Prabhakar, Sudipta Sengupta, and Murari Sridharan. **Data Center TCP (DCTCP)**. In *Proceedings of the ACM SIGCOMM 2010 Conference*, SIGCOMM '10, pages 63–74, New Delhi, India, August 2010. Association for Computing Machinery. 8, 47

[47] Sangtae Ha, Injong Rhee, and Lisong Xu. **CUBIC: A New TCP-Friendly High-Speed TCP Variant**. *ACM SIGOPS Operating Systems Review*, **42**(5):64–74, July 2008. 8, 47

[48] Venkat Arun and Hari Balakrishnan. **Copa: Practical Delay-Based Congestion Control for the Internet**. In *Proceedings of the Applied Networking Research Workshop*, pages 19–19, Montreal QC Canada, July 2018. ACM. 8, 47

[49] Neal Cardwell, Yuchung Cheng, C. Stephen Gunn, Soheil Hassas Yeganeh, and Van Jacobson. **BBR: Congestion-Based Congestion Control**. *Communications of the ACM*, **60**(2):58–66, January 2017. 8, 47

[50] HT Kung. *Traffic Management for High-Speed Networks: Fourth Lecture International Science Lecture Series*. National Academies, 1997. 8

[51] Mario Gerla and Leonard Kleinrock. **Flow Control: A Comparative Survey**. *IEEE Transactions on Communications*, **28**(4):553–574, 1980. 8

[52] Dejan S Milojicic, Vana Kalogeraki, Rajan Lukose, Kiran Nagaraja, Jim Pruyne, Bruno Richard, Sami Rollins, and Zhichen Xu. **Peer-to-Peer Computing**. page 52, 2002. 8, 9, 10

[53] Rüdiger Schollmeier. **A Definition of Peer-to-Peer Networking for the Classification of Peer-to-Peer Architectures and Applications**. In *Proceedings First International Conference on Peer-to-Peer Computing*, pages 101–102. IEEE, 2001. 8

[54] Baochun Li, Yuan Feng, and Bo Li. **Rise and Fall of the Peer-to-Peer Empire**. *Tsinghua Science and Technology*, **17**(1):1–16, February 2012. 8, 10

[55] Philippe Golle, Kevin Leyton-Brown, Ilya Mironov, and Mark Lillibridge. **Incentives for Sharing in Peer-to-Peer Networks**. In *International Workshop on Electronic Commerce*, pages 75–87, 2001. 8, 10, 48

[56] M. Ripeanu. **Peer-to-Peer Architecture Case Study: Gnutella Network**. In *Proceedings First International Conference on Peer-to-Peer Computing*, pages 99–100, Linkoping, Sweden, 2002. IEEE Comput. Soc. 8, 10, 11, 48

[57] M. Izal, Guillaume Urvoy-Keller, Ernst W. Biersack, P. A. Felber, A. Al Hamra, and L. Garcés-Erice. **Dissecting BitTorrent: Five Months in a Torrent's Lifetime**. In David Hutchison, Takeo Kanade, Josef Kittler, Jon M. Kleinberg, Friedemann Mattern, John C. Mitchell, Oscar Nierstrasz, C. Pandu Rangan, Bernhard Steffen, Demetri Terzopoulos, Dough Tygar, Moshe Y. Vardi, Chadi Barakat, and Ian Pratt, editors, *Passive and Active Network Measurement*, **3015**, pages 1–11. Springer Berlin Heidelberg, Berlin, Heidelberg, 2004. 8, 48

[58] Qin Lv, Kai Li, Pei Cao, Scott Shenker, and Edith Cohen. **Search and Replication in Unstructured Peer-to-Peer Networks**. In *In Proceedings of the 16th Annual ACM International Conference on Supercomputing*, 2002. 10

[59] Qin Lv, Sylvia Ratnasamy, and Scott Shenker. **Can Heterogeneity Make Gnutella Scalable?** In Gerhard Goos, Juris Hartmanis, Jan van Leeuwen, Peter Druschel, Frans Kaashoek, and Antony Rowstron, editors, *Peer-to-Peer Systems*, **2429**, pages 94–103. Springer Berlin Heidelberg, Berlin, Heidelberg, 2002. 10, 11, 48

[60] Petar Maymounkov and David Mazières. **Kademlia: A Peer-to-Peer Information System Based on the XOR Metric**. In Peter Druschel, Frans Kaashoek, and Antony Rowstron, editors, *Peer-to-Peer Systems*, Lecture Notes in Computer Science, pages 53–65, Berlin, Heidelberg, 2002. Springer. 10, 30

[61] Ion Stoica, Robert Morris, David Karger, M. Frans Kaashoek, and Hari Balakrishnan. **Chord: A Scalable Peer-to-Peer Lookup Service for Internet Applications**. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '01, pages 149–160, New York, NY, USA, 2001. Association for Computing Machinery. 10

[62] Antony Rowstron and Peter Druschel. **Pastry: Scalable, Decentralized Object Location, and Routing for Large-Scale Peer-to-Peer Systems**. In Rachid Guerraoui, editor, *Middleware 2001*, Lecture Notes in Computer Science, pages 329–350, Berlin, Heidelberg, 2001. Springer. 10

[63] Sylvia Ratnasamy, Paul Francis, Mark Handley, Richard Karp, and Scott Shenker. **A Scalable Content-Addressable Network**. In *Proceedings of the 2001 Conference on Applications, Technologies, Architectures, and Protocols for Computer Communications*, SIGCOMM '01, pages 161–172, New York, NY, USA, August 2001. Association for Computing Machinery. 10

[64] Dongyu Qiu and Rayadurgam Srikant. **Modeling and Performance Analysis of BitTorrent-like Peer-to-Peer Networks**. *ACM SIGCOMM computer communication review*, **34**(4):367–378, 2004. 10, 48

[65] Robert Axelrod. **Effective Choice in the Prisoner's Dilemma**. *Journal of conflict resolution*, **24**(1):3–25, 1980. 10, 48

[66] C. Buragohain, D. Agrawal, and S. Suri. **A Game Theoretic Framework for Incentives in P2P Systems**. In *Proceedings Third International Conference on Peer-to-Peer Computing (P2P2003)*, pages 48–56, Linkoping, Sweden, 2003. IEEE Comput. Soc. 10

[67] Michal Feldman, Kevin Lai, Ion Stoica, and John Chuang. **Robust Incentive Techniques for Peer-to-Peer Networks**. In *Proceedings of the 5th ACM Conference on Electronic Commerce*, EC '04, pages 102–111, New York, NY, USA, 2004. Association for Computing Machinery. 10

[68] Minaxi Gupta, Paul Judge, and Mostafa Ammar. **A Reputation System for Peer-to-Peer Networks**. In *Proceedings of the 13th International Workshop on Network and Operating Systems Support for Digital Audio and Video*, NOSSDAV '03, pages 144–152, New York, NY, USA, 2003. Association for Computing Machinery. 10, 48

[69] Runfang Zhou and Kai Hwang. **PowerTrust: A Robust and Scalable Reputation System for Trusted Peer-to-Peer Computing**. *IEEE Transactions on Parallel and Distributed Systems*, **18**(4):460–473, April 2007. 10, 48

[70] W. Diffie and M. Hellman. **New Directions in Cryptography**. *IEEE Transactions on Information Theory*, **22**(6):644–654, November 1976. 11

[71] Ralph C. Merkle. **Secure Communications over Insecure Channels**. *Communications of the ACM*, **21**(4):294–299, April 1978. 11

[72] Stuart Haber and W. Scott Stornetta. **How to Time-Stamp a Digital Document**. In Alfred J. Menezes and Scott A. Vanstone, editors, *Advances in Cryptology-CRYPTO' 90*, Lecture Notes in Computer Science, pages 437–455, Berlin, Heidelberg, 1991. Springer. 11

[73] David Chaum, Amos Fiat, and Moni Naor. **Untraceable Electronic Cash**. In Shafi Goldwasser, editor, *Advances in Cryptology — CRYPTO' 88*, Lecture Notes in Computer Science, pages 319–327, New York, NY, 1990. Springer. 11

[74] Cynthia Dwork and Moni Naor. **Pricing via Processing or Combatting Junk Mail**. In Ernest F. Brickell, editor, *Advances in Cryptology — CRYPTO' 92*, Lecture Notes in Computer Science, pages 139–147, Berlin, Heidelberg, 1993. Springer. 11, 15

[75] ITU-T Focus Group on Application of Distributed Ledger Technology (FG DLT). **Distributed Ledger Technology Use Cases**. 11

[76] Juri Mattila. **The Blockchain Phenomenon**. *Berkeley Roundtable of the International Economy*, page 16, 2016. 11

[77] Sami Ben Mariem, Pedro Casas, Matteo Romiti, Benoit Donnet, Rainer Stutz, and Bernhard Haslhofer. **All That Glitters Is Not Bitcoin – Unveiling the Centralized Nature of the BTC (IP) Network**. In *NOMS 2020 - 2020 IEEE/IFIP Network Operations and Management Symposium*, pages 1–9, Budapest, Hungary, April 2020. IEEE. 12, 14, 30, 31, 48

[78] Addy Yeow. **Global Bitcoin Nodes Distribution**, 2020 [cited 2020-11-21]. 14, 30

[79] bitfly gmbh. **Clients - Ethernodes.Org - The Ethereum Network & Node Explorer**, 2020 [cited 2020-11-21]. 14, 30

[80] thetangle.org. **Public IOTA Nodes**, 2020 [cited 2020-11-21]. 14

[81] Christian Cachin and Marko Vukolić. **Blockchain Consensus Protocols in the Wild**. *arXiv:1707.01873 [cs]*, July 2017. 15, 16

[82] Y. Xiao, N. Zhang, W. Lou, and Y. T. Hou. **A Survey of Distributed Consensus Protocols for Blockchain Networks**. *IEEE Communications Surveys Tutorials*, **22**(2):1432–1465, Secondquarter 2020. 15, 16

[83] Colin LeMahieu. **Nano: A Feeless Distributed Cryptocurrency Network**. *Nano [Online resource]. URL: https://nano. org/en/whitepaper (date of access: 24.03. 2018)*, 2018. 16, 17

[84] Zahra Ebrahimi, Bryan Routledge, and Ariel Zetlin-Jones. **Getting Blockchain Incentives Right**. Technical report, Tech. rep., Carnegie Mellon University Working Paper, 2019. 16

[85] Andrew Cullen, Pietro Ferraro, William Sanders, Luigi Vigneri, and Robert Shorten. **On Congestion Control for Distributed Ledgers in Adversarial IoT Networks**. *arXiv:2005.07778 [cs]*, May 2020. 17, 22

[86] Qiheng Zhou, Huawei Huang, Zibin Zheng, and Jing Bian. **Solutions to Scalability of Blockchain: A Survey**. *IEEE Access*, **8**:16440–16455, 2020. 17, 48

[87] Kyle Croman, Christian Decker, Ittay Eyal, Adem Efe Gencer, Ari Juels, Ahmed Kosba, Andrew Miller, Prateek Saxena, Elaine Shi, Emin Gün Sirer, Dawn Song, and Roger Wattenhofer. **On Scaling Decentralized Blockchains: (A Position Paper)**. In Jeremy Clark, Sarah Meiklejohn, Peter Y.A. Ryan, Dan Wallach, Michael Brenner, and Kurt Rohloff, editors, *Financial Cryptography and Data Security*, **9604**, pages 106–125. Springer Berlin Heidelberg, Berlin, Heidelberg, 2016. 17, 48

[88] Yinqiu Liu, Kai Qian, Jinlong Yu, Kun Wang, and Lei He. **Effective Scaling of Blockchain beyond Consensus Innovations and Moore's Law**. *arXiv:2001.01865 [cs]*, January 2020. 17

[89] Ana Klimovic, Heiner Litz, and Christos Kozyrakis. **Selecta: Heterogeneous Cloud Storage Configuration for Data Analytics**. In *Proceedings of the 2018 USENIX Conference on Usenix Annual Technical Conference*, USENIX ATC '18, pages 759–773, USA, 2018. USENIX Association. 18

[90] Ronald P. Doyle, Jeffrey S. Chase, Omer M. Asad, Wei Jin, and Amin M. Vahdat. **Model-Based Resource Provisioning in a Web Service Utility**. In *Proceedings of the 4th Conference on USENIX Symposium on Internet Technologies and Systems - Volume 4*, USITS'03, page 5, USA, 2003. USENIX Association. 18

[91] Chenhao Qu, Rodrigo N. Calheiros, and Rajkumar Buyya. **A Reliable and Cost-Efficient Auto-Scaling System for Web Applications Using Heterogeneous Spot Instances**. *Journal of Network and Computer Applications*, **65**(C):167–180, April 2016. 18

[92] Ashraf Mahgoub, Alexander Michaelson Medoff, Rakesh Kumar, Subrata Mitra, Ana Klimovic, Somali Chaterji, and Saurabh Bagchi. **OPTIMUSCLOUD: Heterogeneous Configuration Optimization for Distributed Databases in the Cloud**. In *2020 USENIX Annual Technical Conference (USENIX ATC 20)*, pages 189–203. USENIX Association, July 2020. 18

[93] Alexandru Uta, Alexandru Custura, Dmitry Duplyakin, Ivo Jimenez, Jan Rellermeyer, Carlos Maltzahn, Robert Ricci, and Alexandru Iosup. **Is Big Data Performance Reproducible in Modern Cloud Networks?** In *17th USENIX Symposium on Networked Systems Design and Implementation (NSDI 20)*, pages 513–527, Santa Clara, CA, February 2020. USENIX Association. 18

[94] Devavrat Shah. **Gossip Algorithms**. *Foundations and Trends® in Networking*, **3**(1):1–125, 2007. 22

[95] Juan Benet. **IPFS - Content Addressed, Versioned, P2P File System**. *arXiv:1407.3561 [cs]*, July 2014. 30

[96] Scott A Crosby and Dan S Wallach. **An Analysis of BitTorrent's Two Kademlia-Based DHTs**. page 29, 2007. 30

[97] Shawn Wilkinson, Tome Boshevski, Josh Brandoff, and Vitalik Buterin. *Storj A Peer-to-Peer Cloud Storage Network*. 2014. 30

[98] Mohammad Alizadeh, Shuang Yang, Milad Sharif, Sachin Katti, Nick McKeown, Balaji Prabhakar, and Scott Shenker. **pFabric: Minimal near-Optimal Datacenter Transport**. *ACM SIGCOMM Computer Communication Review*, **43**(4):435–446, August 2013. 47

[99] Elias Rohrer and Florian Tschorsch. **Kadcast: A Structured Approach to Broadcast in Blockchain Networks**. In *Proceedings of the 1st ACM Conference on Advances in Financial Technologies*, pages 199–213, Zurich Switzerland, October 2019. ACM. 48

[100] Gleb Naumenko, Gregory Maxwell, Pieter Wuille, Alexandra Fedorova, and Ivan Beschastnikh. **Erlay: Efficient Transaction Relay for Bitcoin**. In *Proceedings of the 2019 ACM SIGSAC Conference on Computer and Communications Security*, CCS '19, pages 817–831, New York, NY, USA, November 2019. Association for Computing Machinery. 48

[101] Elli Androulaki, Artem Barger, Vita Bortnikov, Christian Cachin, Konstantinos Christidis, Angelo De Caro, David Enyeart, Christopher Ferris, Gennady Laventman, Yacov Manevich, Srinivasan Muralidharan, Chet Murthy, Binh Nguyen, Manish Sethi, Gari Singh,

Keith Smith, Alessandro Sorniotti, Chrysoula Stathakopoulou, Marko Vukolić, Sharon Weed Cocco, and Jason Yellick. **Hyperledger Fabric: A Distributed Operating System for Permissioned Blockchains**. In *Proceedings of the Thirteenth EuroSys Conference*, EuroSys '18, New York, NY, USA, 2018. Association for Computing Machinery. 49

Chapter 7

Appendix

7.1 Artifact Description: Healthor's Simulator

7.1.1 Abstract

This artifact description contains information to set up and run simulations with Healthor as well as reproducing results shown in this thesis. We describe how to obtain the software and data set, the necessary prerequisites to build, and finally, run the OMNeT++-based simulator with the help of the Python plotting framework. The software consists out of two parts: a plotting framework to automatically run sets of experiments, process raw data and plot results, and an event-based simulator that implements Healthor's design as well as the aided and unaided heterogeneity network models.

7.1.2 Artifact Check-list (Meta-information)

- **Algorithm:** Aided, unaided heterogeneity and Healthor.
- **Program:** Healthor simulator (`https://github.com/jonastheis/healthor`).
- **Compilation:** C++11, Python3
- **Data set:** `https://zenodo.org/record/4573698`
- **Experiments:** See file `omnetpp.ini` for available experiment configurations.
- **How much disk space required (approximately)?:** 12.5 GB for all experiments shown in the thesis.
- **Publicly available?:** Source code and data sets are publicly available.
- **Code licenses (if publicly available)?:** Apache License 2.0
- **Data licenses (if publicly available)?:** Apache License 2.0

7.1.3 Description

How to Access

The software can be obtained from GitHub `https://github.com/jonastheis/healthor`:

```
$ git clone https://github.com/jonastheis/healthor.git
```

Software Dependencies

Prerequisites to run Healthor are as follows:

- OMNeT++ 5.6.2
- Python 3
- rocksdb 6.15.*

Data Sets

The results in this thesis can be reproduced by running the experiments from scratch using the simulator, i.e., generating the necessary data. The raw data generated from the experiments shown in this thesis is additionally available on Zenodo `https://zenodo.org/record/4573698`.

7.1.4 Installation

Healthor's framework consists out of the OMNeT++-based simulator and a Python plotting framework for processing the raw data and creating results and plots. This Python framework builds the single entry point to the simulator. It compiles and runs the simulator, shows progress, and runs multiple sets of experiments at once.

```
# set up Python venv
$ cd healthor/plot
$ python3 -m venv venv
$ source venv/bin/activate
$ pip install -r requirements.txt
```

To see all running options and experiments available use

```
$ python thesismain.py --help
```

7.1.5 Experiment Workflow

The plotting framework provides run options for all experiments shown in this thesis. Each one runs a (set of) simulations with OMNeT++ and processes the data resulting in printed output and plots. If run with `-run-simulation false`, the plotting framework expects the raw data to be located in `simulation/results/`. This is useful if the experiments have already been executed before or when using the data set provided on Zenodo. The following command line options are available:

- `microbenchmarks`: run simulation series and plot results of microbenchmarks with 2 different 10 node networks (Section 4.3).
- `macrobenchmarks`: run simulation series and plot results of macrobenchmarks (Section 4.4).
- `attack-analysis`: run simulations and plot results of attack analysis (Section 4.5).
- `centralization-sensitivity-analysis`: run and plot results of centralization score sensitivity analysis (Section 4.4.2).
- `initial-experiment`: run simulation and plot results of the initial experiment aided vs unaided (Chapter 2).
- `processing-variability`: plot processing variability of various cloud hosting provider (Chapter 2).

7.1.6 Evaluation and Expected Results

The plotting framework provides information about the progress and some aggregated results directly on the console. Raw data is saved by the OMNeT++-based simulator to `simulation/results/`. Plots are written to `plot/out/` and `plot/out/t/plots/`.

7.1.7 Experiment Customization

Experiments can be easily customized using the `omnetpp.ini` configuration file for simulation runs. All configurations of simulations conducted in this thesis can be found in this file.