École Polytechnique Fédérale de Lausanne

HotStuff Consensus for Humanitarian Aid Financial Network

by Farida Aclimandos

Master Thesis

Approved by the Examining Committee:

Prof.Bryan Ford
Thesis Advisor

Vincent Graf Narbel
External Expert

Louis-Henri Merino
Pasindu Tennage
Thesis Supervisors

EPFL IC IINFCOM DEDIS
BC 210 (Bâtiment BC)
Station 14
CH-1015 Lausanne

January 19, 2024

Acknowledgments

I would like to thank Louis-Henri Merino and Pasindu Tennage for their constant support and guidance throughout this project.

Lausanne, January 19, 2024 Farida Aclimandos

Abstract

Humanitarian Aid organizations are increasingly interested in issuing digital assets as an alternative to the traditional voucher system. However, the existing infrastructure for digital assets poses significant risks to the privacy and personal safety of beneficiaries.

This thesis explores the concept of a community currency rooted in face-to-face interactions, which, among other properties, enables offline transactions and ensures everlasting privacy. This innovative approach aims to address the limitations and risks associated with current digital asset systems while providing a more secure and private means of transaction for beneficiaries in a humanitarian setting.

Furthermore, delving into the synthesis of the Humanitarian Aid Token (HAT) network with the HotStuff consensus protocol, this thesis evaluates its potential to augment transaction efficacy and reliability in humanitarian aid contexts. It focuses on the integration and performance of the HotStuff algorithm within the HAT network incorporating its specific cryptographic and functional requirements. This research ultimately assesses the system's functionality and effectiveness, contributing to the discussion on employing digital currencies for humanitarian aid delivery.

Contents

Chapter 1

Introduction

In the rapidly transforming landscape of financial systems, the convergence of Community Currencies (CCs) and consensus algorithms has drawn notable interest and research. This thesis ventures into the synthesis of the Humanitarian Aid Token (HAT) network on top of the HotStuff consensus protocol and evaluating the performance of the community-based financial transactions' setup in a simulated humanitarian aid setting.

CCs are localized currency systems tailored to the needs and dynamics of specific communities [3]. They typically take the form of non-interest-bearing physical vouchers or digital tokens, circulated within a specific community. These currencies are not merely economic tools, but social instruments aimed at strengthening local economies, fostering community cohesion, and can therefore enable directed, efficient humanitarian aid.

The work of Jem Bendell, Matthew Slater and Will Ruddick in [1] highlights the Bangla-Pesa initiative by Grassroots Economics to create a community currency in a humanitarian aid setting. Vouchers, either physical or digital, issued and honored by every member of the network function, were used as a form of currency. These vouchers were part of a collaborative credit system, designed to facilitate transactions among community members who might not be part of the credit-issuing community themselves. It allowed the community to trade more and improve its members' lives, yet without additional legal tender.

In parallel, the HotStuff consensus protocol [19] has gained recognition for its robust and efficient mechanism in reaching agreement across distributed systems, particularly within blockchain networks. It is known for its ability to facilitate quick consensus even in environments where trust is minimal, making it a potentially ideal framework for enhancing the integrity and efficiency of CCs.

This thesis is structured to provide a comprehensive understanding of HAT and its network, It is part of an ongoing project at the DEDIS EPFL lab. This thesis' contribution is the implementation and evaluation of this work. It begins with an exposition of HAT, elucidating its design, goals, and the

current state of its network. Subsequently, a presentation of CCs as well as of the HotStuff consensus protocol. Outlining the principles of HotStuff and why it's particularly suited for integration within the HAT network. The thesis then details the implementation of the HAT network on top of the HotStuff algorithm all while meeting HAT's needs and specifications.

Furthermore, this research does not merely present a theoretical framework; it delves into the practical application and empirical evaluation of this integration. It maps out the journey from concept to implementation, providing insights into the challenges encountered and the solutions devised. The concluding section is dedicated to a performance evaluation of this tailored adaptation, providing insights into its efficacy and functionality.

By marrying the local empowerment potential of CCs with the technological robustness of consensus algorithms, this project contributes to the field, providing a secure and efficient solution for humanitarian aid distribution.

Chapter 2

Background

This section provides a comprehensive overview of CCs and consensus algorithms to establish a foundational understanding of the project's scope. In the evolving landscape of humanitarian aid and financial ecosystems, the convergence of CCs and consensus algorithms has garnered significant attention, setting the stage for the proposed integration of the HotStuff consensus algorithm into the HAT system.

2.1 Community Currencies

Community currencies (CCs) [8] are forms of currency used within a specific community and are most commonly issued by a central party. They are circulated and backed by interest groups, foundations or private companies to the difference of conventional fiat currencies which are backed by a central bank or monetary authority. The central party may guarantee redemption of currency units for government-backed fiat currency in order to engender trust in the currency. The usual setup is that they can only be redeemable for goods and services at participating merchants.

The most common use of community currencies is as a means of encouraging trade within a community. They have emerged as a response to the growing inequality and social exclusion in modern societies. They seek to promote a sense of community and support small and medium enterprises (SMEs) by providing an alternative to traditional fiat currencies.

Community currencies foster economic resilience and social cohesion and are used in several communities worldwide. This economic ecosystem fosters an environment where diverse skills and services flourish, independent of the pressures of globalized markets. An important aspect to consider is the impact community currencies can have in a humanitarian aid context.

2.1.1 Key Characteristics

To understand the dynamics of CCs it is important to delineate their key characteristics.

Acceptance. Community Currencies are generally issued and managed by local organizations whereas, national currencies are issued, managed and regulated by a central bank and the overseeing government. CCs often face natural barriers to being used outside their designated vendors. This ensures that the currency circulates predominantly within the community it serves, reinforcing the intended local economic impact. National currencies are always legal tenders while CCs are merely an agreement between the concerned parties.

Purpose. CCs often address a very specific social or economic need while national currencies aim to be a store of value.

Stability. CCs are used in times of turmoil because they can then offer better stability than national currencies. For example the scrip movement in the United States during the Great Depression [6].

2.1.2 Limitations

The perceived limitations of CCs, such as limited reach and the complex journey from local tokens to national currency, are not inherently problematic within a humanitarian aid context. In these settings, Community Currencies are typically deployed to complement traditional aid methods, addressing various challenges and bolstering the efficacy of humanitarian assistance. These currencies notably enhance the dignity and choice of beneficiaries, empowering them to select the goods or services they most require, rather than being constrained to pre-determined provisions. This stands in contrast to solutions like vouchers, which may limit choice and agency.

Furthermore, with the proliferation of mobile technology, digital CCs are increasingly prevalent. These digital formats facilitate more straightforward distribution, tracking, and management, making them particularly suited for the dynamic and often urgent nature of humanitarian aid. Such advancements in technology not only extend the reach and potential of CCs but also mitigate some traditional limitations associated with their physical counterparts, illustrating that the constraints of CCs do not necessarily hinder their application in humanitarian settings.

2.1.3 Examples

Examining previous examples and case studies of CCs provides valuable insights into their implementation and the impact they have, particularly in the humanitarian sector.

A notable example is Grassroots Economics in Kenya, highlighted in a study published in Frontiers in Blockchain [8]. This organization has been pivotal in establishing Community Inclusion Currencies (CICs) such as Sarafu, a blockchain-based currency, to facilitate trade and credit among people with limited access to traditional banking. They used a Collaborative Credit System (CCS) in which members issue interest-free credit to each other. This is similar to how most national currencies are created, yet it is done peer-to-peer, without the involvement of banks.

In the context of Europe, the Community Currency Innovation Agency (CCIA) provides a legal and compliance toolkit for CCs, supporting similar initiatives.

Various examples of CCs highlight their diverse applications [3]. The Lambeth Pound in London's Lambeth Borough and the Bristol Pound in the UK are local currency initiatives aimed at promoting spending within local businesses, thus strengthening local economies by encouraging money circulation within the community. In the Netherlands, the Makkie currency used in the Indische Buurt neighborhood operates on a time-based system, encouraging residents to earn Makkies through community service, which can then be spent locally. This concept parallels the Spice Time Credits in the UK, fostering community development and social inclusion. TradeQoin in the Netherlands represents a business-to-business trade network, designed to assist SMEs in trading goods and services without relying heavily on traditional cash transactions.

At its core, money is a social contract legitimized by the acceptance of its users. CCs, particularly in settings like Kenya, embody this concept, offering a more participatory and sustainable model of humanitarian assistance. This shift underscores the transformative potential of CCs in fostering inclusive and resilient local economies.

2.2 Consensus Algorithms

A consensus algorithm is a mechanism used in distributed systems, particularly in blockchain technology. It is designed to achieve agreement on a single state among distributed processes, some of which can be unreliable. Despite the absence of a central authority, blockchain networks remain secure and verified due to consensus algorithms. A blockchain also relies on these consensus protocols to maintain its key features of immutability, privacy, security, and transparency.

In our project, we have utilized a permissioned blockchain. This type of distributed ledger is not open to the public, access is restricted to users who have been granted specific permissions. Participants in this system are required to identify themselves through certificates or other digital

means and are limited to actions authorized by the community leaders.

Consensus, in this context, is a procedure that allows a set of replicas or peers to reach a common agreement about the state of the distributed ledger. There are various consensus algorithms used to achieve this, each with their unique characteristics and applications:

Proof of Work (PoW). PoW was first introduced in Bitcoin as a way to secure the network and prevent double-spending. PoW requires miners to solve complex mathematical puzzles known as hashes. The first to solve the puzzle adds a new block to the blockchain, earning cryptocurrency rewards. The algorithm's security comes from the fact that the hash is difficult to solve which makes it expensive for an attacker to try to overtake the network. While highly secure, PoW is energy-intensive and slow, making it less efficient for some applications.

Proof of Stake (PoS). PoS relies on validators who hold a certain amount of cryptocurrency to validate transactions and add new blocks to the chain. This method is more energy-efficient than PoW and offers quicker transaction validations.

Delegated Proof of Stake (DPoS). DPoS is a variation of PoS, where a few elected delegates are responsible for validating transactions and adding new blocks to the chain. This system improves efficiency but can lead to centralization if a small group of delegates controls the network.

Proof of Authority (PoA). PoA relies on a pre-approved group of trusted validators instead of a decentralized network of nodes. This consensus algorithm is faster and more efficient than others such as PoW or PoS. It allows for a more controlled and centralized approach to validation, which may be more desirable in certain contexts such as permissioned blockchains.

Byzantine Fault-Tolerance (BFT). BFT is a computer science concept that refers to the ability of a computing system's ability to function correctly even when enduring arbitrary failures or malicious behavior of its components. In a blockchain context, this means that if two thirds of the nodes agree of the validity of a transaction it is added to the chain. This is crucial in a context where reliability and fault-tolerance are priorities. BFT is often used in private or enterprise blockchain networks where participants are known and trusted.

Practical Byzantine Fault-Tolerance (PBFT). This consensus is an optimized version of the BFT algorithm to provide a high level of fault-tolerance in distributed systems. It reduces the computational complexity and improves the system's performance, making it especially suitable for permissioned blockchains. The PBFT algorithm requires a certain number of nodes in order for a

transaction to be added to the chain, this number being $f = (n-1)/3$ where n is the total number of nodes in the network and f is the maximum number of faulty nodes.

2.2.1 HotStuff

In this section, we delve into the specifics of the HotStuff consensus algorithm, a protocol that is recognized as a leader-based BFT replication protocol. It is tailored for the partially synchronous model, which means adapts to changing network conditions where communication delays and network partitions may occur [10]. Its unique approach seamlessly integrates classical BFT foundations with blockchain technology [19].

The resilience of HotStuff in a partially synchronous network is further bolstered by its Byzantine fault-tolerance. The system operates under the assumption that out of a total of $3f+1$ governing committee members, up to f members may be compromised or unavailable. This threshold is critical in maintaining the integrity and robustness of the system even under conditions where certain committee members are not functioning as expected.

Thus, HotStuff provides a reliable and secure framework for reaching consensus, even in the face of unpredictable network behavior and potential internal faults. That is why it is particularly suited for an application such as the HAT system.

System Model

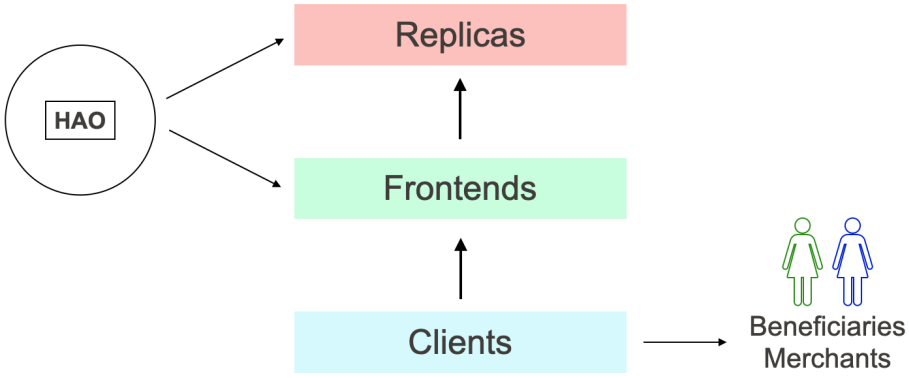


Figure 2.1: HotStuff System Model

In the depicted system model of HotStuff, as illustrated in Figure 2.1, we can see that there are three primary entities. These include the clients, who, within the HAT framework, are identified as the beneficiaries and merchants. The two other critical components are the front-ends and the replicas, both of which are within the Humanitarian Aid Organization (HAO). In this system,

clients initiate interactions by sending their commands to the front-ends. These front-ends are then responsible for aggregating these commands into batches. Subsequently, the replicas, proceed to processing these batches, thereby ensuring the smooth operation of the HAT system.

Now let's take a closer look at a HotStuff replica:

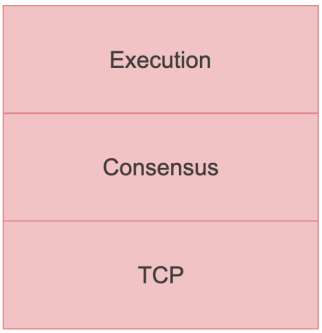


Figure 2.2: HotStuff Replica

In the figure illustrating the model of a HotStuff replica, as shown in Figure 2.2, we observe a structured composition consisting of three distinct layers:

The TCP Layer. The regular transport layer using TCP as transmission protocol, which handles the reception and transmission of packets over the network. It forms the basis for network communication, ensuring that data packets are sent and received reliably between nodes in the system.

The Consensus Layer. Central to the HotStuff protocol, this layer is where the consensus algorithm is executed, allowing the replicas to reach an agreement. HotStuff adopts a three-phase core process for achieving consensus, as detailed in [19]. The first phase of this process is centered around ensuring the uniqueness of proposals through the use of a Quorum Certificate (QC) [15]. The QC is a cryptographic proof ensuring that a proposal has been accepted by a sufficient number of replicas, preventing conflicting proposals from being accepted.

In the second phase, the protocol enables a new leader to garner support for a safe proposal. This is accomplished by relaying information from $(n - f)$ replicas, each reporting its own highest QC or vote. This information allows the leader to construct a proposal that is likely to be accepted by the replicas, ensuring the safety of the consensus process.

The third and final phase, is crucial to the transition of leadership. It allows a new leader to simply pick the highest QC it knows of, a strategy that not only simplifies the leader replacement protocol but also significantly mitigates the communication complexities often associated with consensus algorithms. This approach effectively alleviates the quadratic communication bottleneck,

enhancing the overall efficiency of the protocol.

Additionally, HotStuff introduces the "safety-voting" mechanism, where participants vote on the safety of a block (assurance that the block, once committed, cannot be replaced or reverted without violating the protocol's rules) before voting on its validity. This ensures safety in the consensus process. Moreover, the protocol adeptly manages view changes, which allows the system to promptly recover from leader failures.

The Execution Layer. Being a chain replication algorithm, the default operation mode is a serially threaded system. This implies that all tasks, particularly verification tasks, are processed sequentially. The inherent nature of a serially threaded system in the standard HotStuff framework, could lead to operational halts, especially when faced with the need to verify a large volume of transactions or data. This can lead to processing inefficiencies due to the accumulation of tasks in a single thread. To address these challenges within the context of the HAT project, we have implemented an execution layer that better aligns with our specific requirements and specifications. The details about our approach can be found in the chapter 4 of this report discussing our implementation.

Properties

HotStuff, distinguished for its responsiveness, is adept at guiding the consensus process at a pace congruent with the actual network delay. This capability is achieved through a design that enables the leader to reach consensus with a communication complexity linear in the number of replicas, a feature uniquely absent in other partially synchronous BFT replication protocols.

In addition to this, HotStuff is notable for its robust liveness property. This property ensures that the system continues making progress, even amid failures or delays, with the assurance that each client request is eventually committed. The protocol assumes that client requests are repeatedly proposed by replicas until successfully committed. Complementing this, HotStuff introduces the Pacemaker component, designed specifically to uphold liveness. It extends the graph with new nodes, allowing the protocol to progress and reach consensus even in the presence of failures or delay [15].

Moreover, HotStuff ensures safety, a critical property that guarantees both the finality and correctness of the decisions made by the consensus protocol. This ensures that all honest participants in the network reach agreement on the same sequence of blocks.

The protocol's ability to guarantee both liveness and safety under Byzantine conditions makes it an effective solution for environments demanding high fault-tolerance and operational integrity.

Practicality for Blockchain

HotStuff is noteworthy for its inclusive framework which allows for the expression of other well-known protocols such as DLS, PBFT, Tendermint, and Caspe in a common framework. A key advancement of HotStuff is its linear communication complexity, a notable enhancement compared to certain existing BFT replication protocols like BFT-SMaRt, which are characterized by quadratic communication complexity. This reduction in communication overhead significantly improves HotStuff's efficiency and scalability making it a more effective choice for systems where rapid and reliable consensus is crucial. This underscores HotStuff's innovative contribution to the field of distributed computing and blockchain technology.

Chapter 3

Humanitarian Aid Token (HAT)

The goal behind creating a Humanitarian Aid Token is to enable the International Committee of the Red Cross (ICRC) to deliver monetary assistance to beneficiaries at scale. The idea is that the community driven financial network would permit the beneficiaries to purchase their essential goods while permitting the ICRC to support the local economy. This is part of an ongoing project at the DEDIS EPFL lab.

3.1 System Model

This section details the HAT system model, the system properties, the system workflow as well as the threat model.

3.1.1 System Properties

The system is specifically designed to focus on achieving a set of distinct and critical properties further detailed below.

Everlasting Privacy. A paramount property is everlasting privacy, ensuring that members' financial records remain confidential, similarly to the privacy offered by physical cash. This ensures an individual's financial history is protected indefinitely unless disclosure is mandated by law.

Delay-Tolerant Transactions. Another critical property is delay-tolerant transactions, which, akin to physical cash transactions, allows sellers to accept payments with a reasonable level of risk and convenience, even in scenarios where participants are not online.

Coercion-Detection. The system also incorporates coercion-detection, enabling members to discreetly inform the governing committee of their situation if they are under duress.

Currency Management. Currency Management is another crucial aspect, where a designated governing committee has the authority to mint and burn currency. This committee is also responsible for implementing various financial policies. Such as capital controls, which regulate the conversion of community currency to reserve assets. Or demurrage to discourage hoarding by periodically reducing the balance amount of each member, a negative interest. And potentially distributing a universal basic income to all or specific members to promote financial stability and equality.

Mutual Accountability. Lastly, the system upholds Mutual Accountability, allowing for a reciprocal oversight mechanism where the governing committee can hold members accountable for any misuse of the system, and conversely, members can hold the committee responsible for any mismanagement or malpractice such as fraud or embezzlement.

These properties collectively ensure that the system operates effectively, securely, and equitably.

3.1.2 Actors

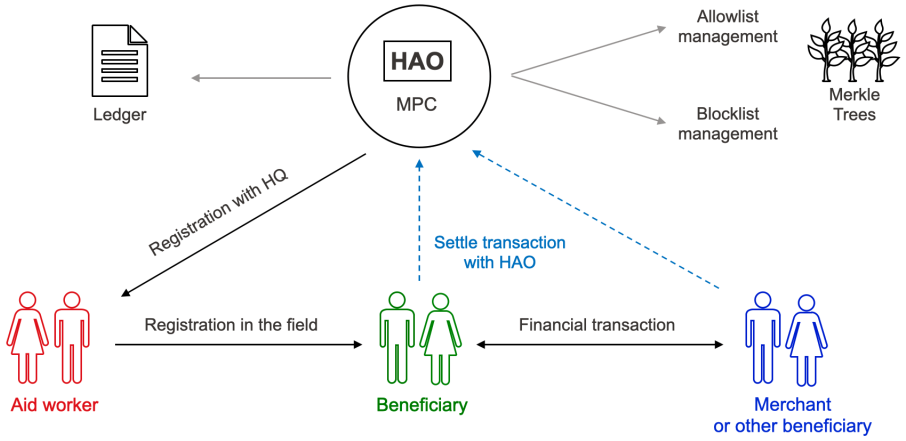


Figure 3.1: HAT system model

As can be seen in the graph of the HAT network, Figure 3.1, there are two categories of actors. Firstly, the governing committee members and secondly, the community members which include the beneficiaries, the merchants and the humanitarian aid workers.

Governing Committee. The Humanitarian Aid Organization (HAO) or Governing Committee is denoted $\mathscr{C} = \{C_1, C_2, ..., C_{n_\mathscr{C}}\}$, $n_\mathscr{C}$ being the number of committee members. They are in charge of minting and burning the currency, processing transactions as well as evaluating applications from prospective members. The HAO also holds community members accountable in case any abuse is detected, investigate cases of detected coercion. They oversee the entirety of the monetary policy.

Community Members. The set of community members consists of the humanitarian aid workers, which are the initial members of the community currency, the ICRC-approved beneficiaries and the ICRC-approved merchants. The aid workers and ICRC-approved beneficiary can transact with an ICRC-approved merchant as well as with another ICRC-approved beneficiary.

Prospective Members. Prospective members designate merchants or beneficiaries who have not completed the enrollment phase which means they have not yet been approved by the HAO. The enrollment process for prospective members is done in-person.

3.1.3 Procedure

There are three steps to the community currency's procedure: Setup, Enrollment and Transaction.

Setup. During Setup, the governing committee $\mathscr{C}$ generates their respective secret keys in-person and establishes the policies for the currency. They also designate themselves as the initial members of the community currency.

Enrollment. In order for a prospective member M to enroll, an existing community member M_e has to vouch for them. M then has to provide the committee with all the documents necessary for their application and identification.

Once verified and enrolled, the ICRC-approved beneficiaries can transact with ICRC-approved merchants, other beneficiaries or vouch for unregistered parties.

To detect coercion, members register both a normal and a panic PIN during Enrollment. If the latter is used during a transaction, it alerts the committee that the member is under duress.

Transaction. In this system, a buyer can initiate a transaction either by scanning a QR code displayed on the seller's device or by inserting their smart card into the seller's payment terminal, finalizing the transaction by entering their PIN.

For online transactions, the recipient's device might prompt the sender to enter a prepaid

amount for future transactions. However, in the case of an offline transaction, where both member devices are disconnected, and no prepaid amount has been set, the governing committee's policies come into play to protect the recipient against potential fraud. If these policies are violated, the loss is borne by the recipient, much like how the issuer of physical cash makes efforts to prevent counterfeit bills but ultimately, if a recipient accepts counterfeit currency, the loss is theirs. Before any transaction is initiated, it is crucial to ensure that the other person is on the allowlist in the HAT system and is not on the blocklist. Both verifications are required as they are not updated at the same frequency.

This system allows two parties to transact seamlessly without internet connectivity, offering the same advantages as cash. However, only merchants approved by the ICRC can exchange tokens for fiat currencies. This exchange process is contingent on the approval of both the ICRC and the involved financial institution.

Once online, transactions are settled with the HAO, which is responsible for processing the transactions. This is where the HotStuff consensus protocol comes into play within the Multi-Party Computation (MPC) framework, facilitating the generation of a secure and transparent ledger.

This overall Transaction approach underscores the concept of offline transactions, aiming to establish a community currency rooted in face-to-face interactions. It not only enables transactions in the absence of connectivity but also ensures everlasting financial privacy akin to physical cash. Simultaneously, it provides stronger transparency and accountability guarantees than cash.

3.2 Threat model

The system is designed to withstand attacks from two types of non-colluding adversaries, reflecting real-world threats. Adversaries operate with the intent to expose the contents of transactions and disrupt the financial network.

The first adversary possesses comprehensive visibility over the internet communications between users and is computationally unbounded. This adversary is capable of breaking hardness assumptions such as the Discrete Logarithm (DL) or Lattice-based cryptography but refrains from revealing this capability to avoid public detection.

The system's design ensures that even if DL is compromised, overall security is maintained. However, the public auditability property would be forfeited, leaving the system to rely solely on honest MPC. This global actor seeks to either gain financial insights into targeted communities, to compromise privacy or to disrupt the financial network's integrity while concealing their most powerful tools.

The second adversary, in contrast, is computationally bounded but possesses the ability to interact directly with community members. Their goal is to coerce victims into surrendering access to their accounts or to perform transactions on their behalf. This adversary represents a more personal and immediate threat, using social engineering and other forms of coercion to achieve

their ends.

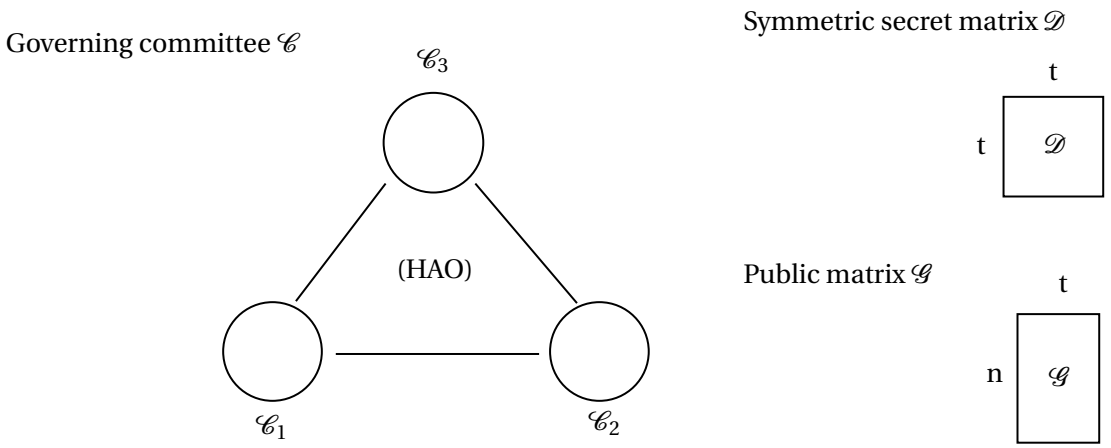
3.3 Cryptographic scheme

This section presents the cryptographic scheme.

Setup. In the initial Setup phase, the governing committee, denoted as $\mathscr{C}$, is responsible for creating a symmetric secret matrix $\mathbb{D}$, which is secret shared among the committee members. Additionally, the committee generates a public key for every community member, derived from a public matrix $\mathbb{G}$, and establishes an initial blocklist.

A trusted entity, denoted as $\mathbb{T}$, is assumed to take the place of a distributed key generation protocol, establishing long-lived secret communication keys. This entity generates a secret key for each pair of committee members using AEAD encryption and selects a prime number p. It then generates the symmetric secret matrix $\mathbb{D}$, with dimensions $t \times t$, over the finite field $\mathbb{Z}_p$. This matrix is secret shared among the committee members, indicated as $[\mathbb{D}]^C$.

Furthermore, $\mathbb{T}$ pre-generates n public vectors pk_i of size t over $\mathbb{Z}_q$ (where $q >> p$), each corresponding to a prospective community member, and distributes them to all committee members. These vectors serve as the community member's public keys.



The individual elements of the secret matrix $\mathscr{D}$ are distributed among the committee members via MPC. Each row of the public matrix $\mathscr{G}$ represents a community member's public key, with n signifying the number of community members.

Every committee member j calculates their share for each community member's secret key by performing a dot product of each row of the public matrix with their share of the secret matrix, culminating in their corresponding matrix $\mathbb{k}$. In other words, each committee member j computes

their share of the private vector $[sk]_j^n$ for each community member i by multiplying matrix $[S]_j^C$ with the member's public key pk_i : $[sk_i]_j^C = \mathscr{G}.pk_i$. These private vectors represent the private keys for community members.

The committee members also pre-generating communication identifiers (CIDs) that will be used for communication between the committee and community members. These CIDs are generated under multi-party computation (MPC) as $CID_{i,C} = PRF_{[sk_i]^C}(CID_{i,c-1}) \forall c \in \{0, N_{CID}\}$, where N_{CID} is an estimate of the total number of communications expected between the community member and the committee. The CIDs can be periodically generated, so underestimating N_{CID} is not an issue.

Additionally, the committee also sets up an initial blocklist using a Merkle Patricia Tree, that initially, does not contain any elements.

In the end, each community member receives, from the governing committee, their public key and all the shares constituting their private key. Each community member is equipped to reconstruct their private key sk_i from each share $[sk_i]_j^C$ using Lagrange Interpolation.

Finally, the committee provides each community member with a proof affirming they are on the allowlist.

Enrollment. The enrollment process for a potential community member, denoted as M, involves submitting an application to the committee members using a CID obtained from an existing community member M_c. The goal is to acquire an unassigned public key, its corresponding private key, and proofs for both the allowlist and blocklist.

Enrollment must be conducted in-person with a trusted and already enrolled community member and can subsequently be completed using their personal devices. Initially, M receives an unused CID from M_c. Then M compiles all necessary information for the committee application, encrypts it using AEAD with the CID, and forwards it to the committee. Upon the committee's approval, M is then granted a public key, the matching private key, and two proofs: one confirming their inclusion on the allowlist and another for the blocklist.

Transaction. When a transaction between a buyer, noted B and a seller S takes place, the seller S provides the buyer B with the amount a, their public key pk_s and a proof p_s that they are not on the blocklist θ.

The buyer then verifies the amount and the validity of the proof p_S using their local blocklist root hash. They enter their PIN and then compute the shared secret key between B and S:

$$K_{BS} = sk_B \cdot pk_S = pk_B \cdot sk_S \tag{3.1}$$

B computes two approval signatures, one for the committee:

$$\sigma_C = PRF_{sk_B}(a \| pk_S \| PIN) \tag{3.2}$$

And the second for the seller:

$$\sigma_S = Sig.Sign_{sk_B}(a \| pk_S) \tag{3.3}$$

The buyer then computes a Pederson commitment:

$$\tau = g^a h^r; \quad r \leftarrow \mathbb{Z}_p \tag{3.4}$$

Finally, the buyer transmits the following to the committee:

$$AEAD_{CID_{b,c}}[AEAD_{k_{BS}}(a, \sigma_C, \sigma_S, \tau, r), pk_B, pk_S], CID, PRF(CID) \tag{3.5}$$

Chapter 4

Implementation

4.1 HotStuff in the HAT network

In this section, we will delve into the implementation of HAT on top of the HotStuff algorithm and how we met the specific requirements and properties of the system. In the HAT system model in Figure 4.1 we can clearly see where HotStuff fits within the HAT network.

We have tailored the execution layer on top of the HotStuff protocol to better align with the unique requirements of the HAT network, aiming to boost its efficiency. The standard HotStuff model, being a chain replication algorithm, operates on a serially threaded system. However, this setup can lead to operational halts when handling a high volume of transactions or data. To address these limitations, our implementation diverges from the conventional single-threaded approach in order to address the potential bottlenecks and synchronization issues, which can result in processing inefficiencies.

Our solution incorporates a multithreading strategy, which, while solving some the challenges of the original model, introduced new complexities in synchronization. Such as managing multiple threads concurrently without causing data corruption or other synchronization-related issues.

To effectively mitigate these risks, our implementation strategy involved separating tasks into different channels. This design ensures that the handling of client requests, here transactions, did not bottleneck the overall consensus mechanism. By distributing tasks across multiple channels, our system achieves parallelization of operations, thereby enhancing throughput and reducing delay risks.

The success of this multichannel approach is heavily dependent on the availability of sufficient memory resources. Adequate memory is essential for efficiently allocating and managing tasks across different channels. With the necessary memory resources, our modified execution layer in

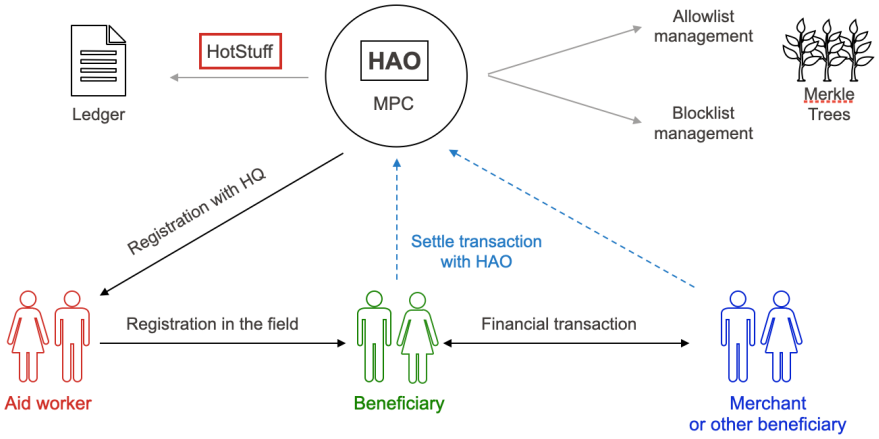


Figure 4.1: Hotstuff within the HAT network

the HotStuff protocol elevates the performance and scalability of the HotStuff protocol within the HAT network.

A critical consideration in our implementation is the timing of the execution layer processing. Processing before consensus allows for the validations of values and transactions before they are committed to the chain but, no matter which multichannel approach we take, it delays the finalization of the consensus process. Conversely, validating client requests post-commitment risks introducing unverified data onto the chain potentially leading to a proliferation of invalid entries.

To accurately simulate the behavior of the HAT framework, it is crucial to consider the potential for high bursts in traffic. Such surges could arise from various scenarios, such as a multitude of offline transactions coming online simultaneously when connectivity is restored, or an increase in client transactions coinciding with the arrival of aid.

Through our analysis, we've determined that predicting a specific behavioral pattern for humanitarian aid is challenging due to its heavy dependence on environmental factors rather than consistent usage patterns. Consequently, each case tends to be unique, precluding the establishment of a universal pattern.

Therefore, in our performance evaluation, we focus on testing the system by examining its response to extreme cases and burst traffic scenarios. This approach allows us to assess the system's robustness and scalability under high-load conditions, which are critical for understanding its real-world efficacy. As a result of these considerations, we have adapted HotStuff client request generation mechanism. This adaptation is designed to better mimic the unpredictable and varied nature of traffic in the HAT system.

4.2 Implementing the HAT network

In the development of the HAT network, there are several crucial components at the core of our implementation. These components include instances designated for members of the governing committee, the HAT community members and the transaction processing framework. Each plays a vital role in the overall functionality and efficiency of the HAT network.

Our implementation approach is structured around three main steps: **Setup**, **Enrollment** and **Transaction**. Central to these steps is the integration of cryptographic techniques and MPC. The incorporation of these elements is critical in ensuring the security, integrity, and privacy of transactions within the HAT network.

During the **Setup** phase, our initial task involves simulating the trusted entity and generating all necessary cryptographic elements. This foundational step lays the groundwork for a secure and reliable network infrastructure. Moving to the **Enrollment** phase, for the purposes of our performance evaluation, we proceed under the assumption that participants are already enrolled in the network. This assumption is instrumental in streamlining our implementation process, allowing us to concentrate on key operational aspects such as transaction processing and other critical functionalities. The **Transaction** stage is particularly significant for evaluating the network's performance in scenarios reflective of real-world use.

In our implementation, the prioritization of the development and integration of components most relevant to the performance evaluation of the HAT network has been a strategic decision. This focus ensures that we can rigorously test and assess the network's efficiency, scalability, and reliability under various conditions, thereby providing a solid foundation for further development and optimization of the HAT system.

Chapter 5

Performance evaluation

We evaluate the HAT system to address the following key questions.

Firstly we want to evaluate the performance of adding an execution layer. For that, we assess the baseline system performance under conditions where the client requests consist of random bytes and have a low payload. They therefore do not require any further processing. This allows the measurement of the raw throughput and latency of the HotStuff algorithm without the influence of the HAT system components. Following this, we integrate the HAT transaction architecture into the system. This phase involves simple transaction processing, which adds the complexity of a simple execution layer to the requests and, consequently, allows us to evaluate the impact of the HAT-specific transactional execution architecture on the system's performance.

Secondly we want to evaluate the influence of client requests generation patterns, namely how the performance varies when there is a sudden burst of traffic rather than a fixed Poisson distribution in the open-loop model [11] with a client arrival rate fixed at 100 requests per second. For that we test those two scenarios within experimental setups with varying numbers of replicas and clients in order to show how the system scales and try and get closer to a real life application.

Finally, we want to know the performance cost to adding our cryptographic scheme, which means a higher client request size as well as a slightly more complicated execution layer. This addition is expected to introduce additional processing requirements, thereby providing insights into how HAT cryptographic operations influence the overall performance of the system in terms of throughput and latency.

Through these sequential steps of increasing complexity, we aim to comprehensively understand and document the scalability and efficiency of the HAT network. This analysis is crucial for identifying potential bottlenecks and areas for optimization, ensuring the robustness and reliability of the system in real-world applications.

5.1 Experimental Configuration

We have implemented, using go version go1.21.1 [16] in 12420 lines of code as counted by CLOC [4], the HAT network on top of an existing implementation of the HotStuff protocol. We use the standard Go network library and TCP [17] for reliable point-to-point links between replicas.

This section is dedicated to examining and comparing various aspects of the HAT system's performance, specifically focusing on throughput and latency. The throughput in this setup is defined as the number of clients requests per second for which a response was received. Furthermore, the latency is defined as the difference between the time at which a client sends a client sends a request and the time at which a corresponding response is received. The values shown are averaged values of 3 runs of each experiment setup. Please note that when interpreting the results, in the graphs for example, what is designated by a packet is a client request.

We use virtual machines in the DeterLab framework [5], the nodes are configured with at least 4 GB of RAM and at least 4 CPU cores. The network uses IPv4 addressing and static routing, ensuring that all nodes can communicate with each other.

5.2 Baseline Performance

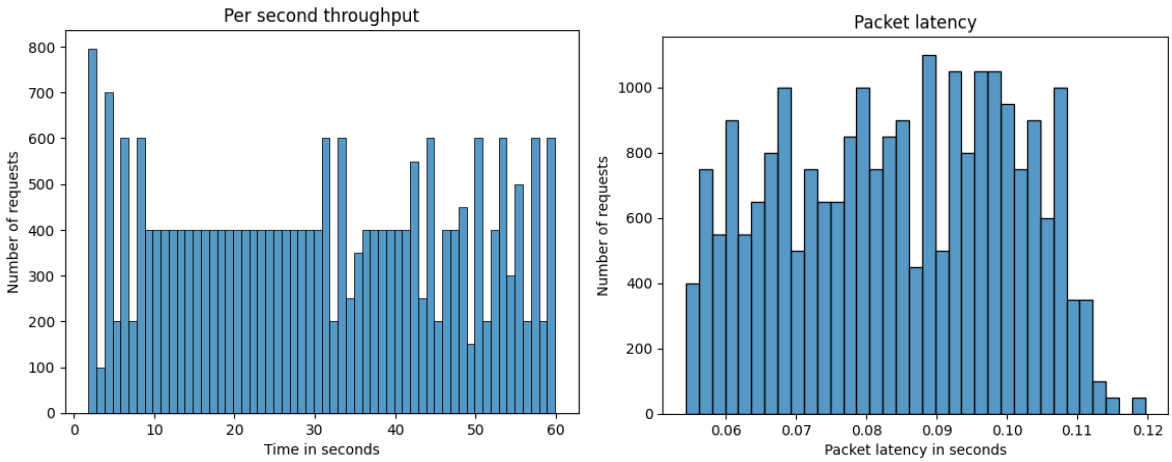
In order to evaluate the HotStuff algorithm's baseline performance, we evaluate a case with minimal complexity, where client requests have a payload of 8 bytes and there is no additional processing required.

5.2.1 Fixed Poisson Arrival Rate

In our analysis, we conducted experiments using two distinct configurations of machines to evaluate the performance of our system under varying conditions all while focusing on throughput and latency.

The first configuration utilized a total of 8 machines, comprising 4 replicas and 4 clients. Client requests were modeled to follow a fixed Poisson arrival rate. The throughput, as depicted in the throughput graph (Figure 5.1a), exhibited a nearly constant rate of around 400 requests per second. This rate, while steady, does not represent the system's maximum capacity, as will be evident in scenarios featuring burst traffic. The chosen Poisson distribution in this setup resulted in a moderately low packet throughput. The system displayed a median latency of 84,823 microseconds, with a 90th percentile latency reaching 105,301 microseconds (Figure 5.1b).

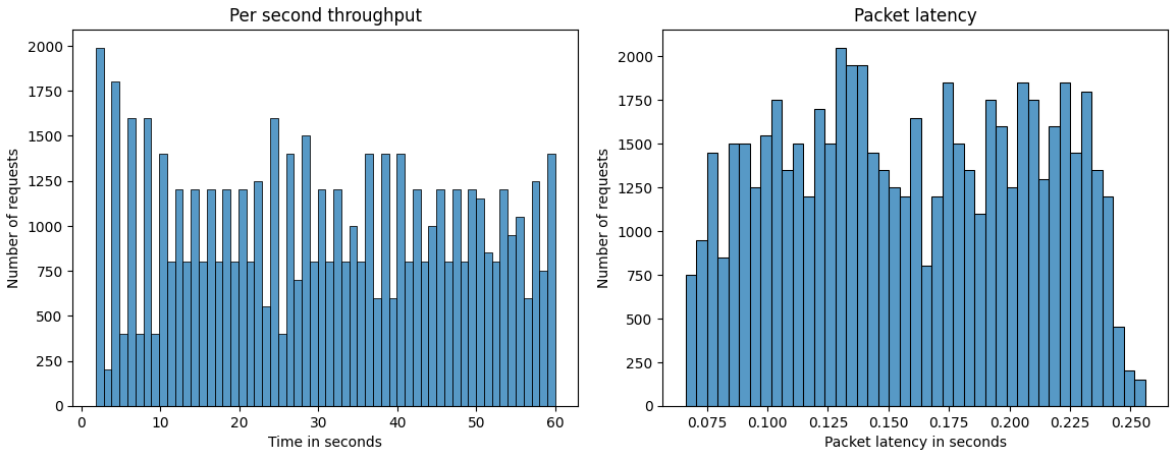
To further explore the scalability and performance of our system, a second configuration was



(a) Throughput per second

(b) Packet latency baseline

Figure 5.1: Baseline performance with fixed Poisson arrival rate, setup with 8 machines



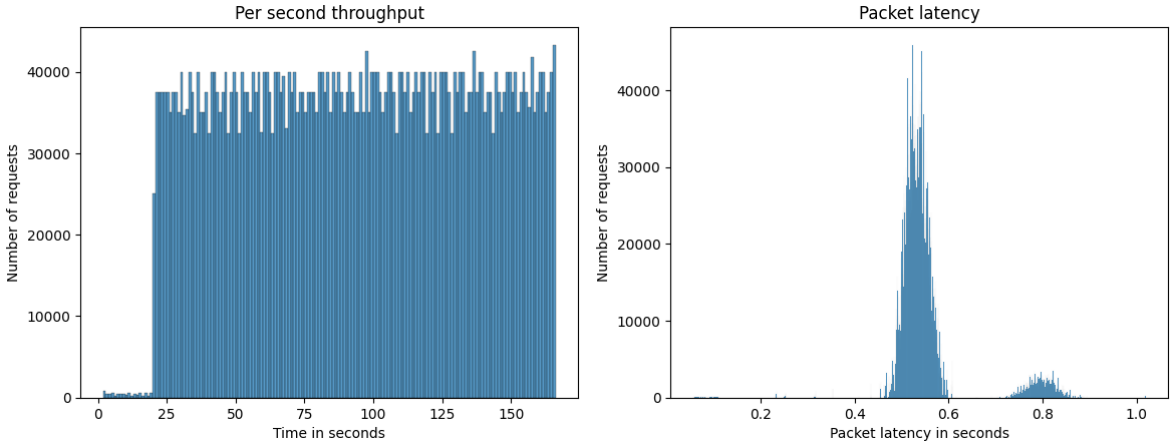
(a) Throughput per second

(b) Packet latency baseline

Figure 5.2: Baseline performance with fixed Poisson arrival rate, setup with 20 machines

implemented, involving an increased scale of 20 machines, split between 10 replicas and 10 clients. Similar to the first setup, the client requests adhered to a Poisson arrival rate. The throughput, as indicated in the graph (Figure 5.2a), was around 990 requests per second, remaining relatively constant. Similar to the previous setup, this throughput does not represent the peak capacity of the system, as will be indicated by the results obtained under burst traffic conditions. The increased throughput in this scenario is primarily due to the greater number of messages being processed on the network, a result of the higher number of machines involved. The latency metrics in the 20-machine configuration showed an increase, with a median latency of approximately 157,690 microseconds and a 90th percentile latency of about 227,389 microseconds (Figure 5.2b).

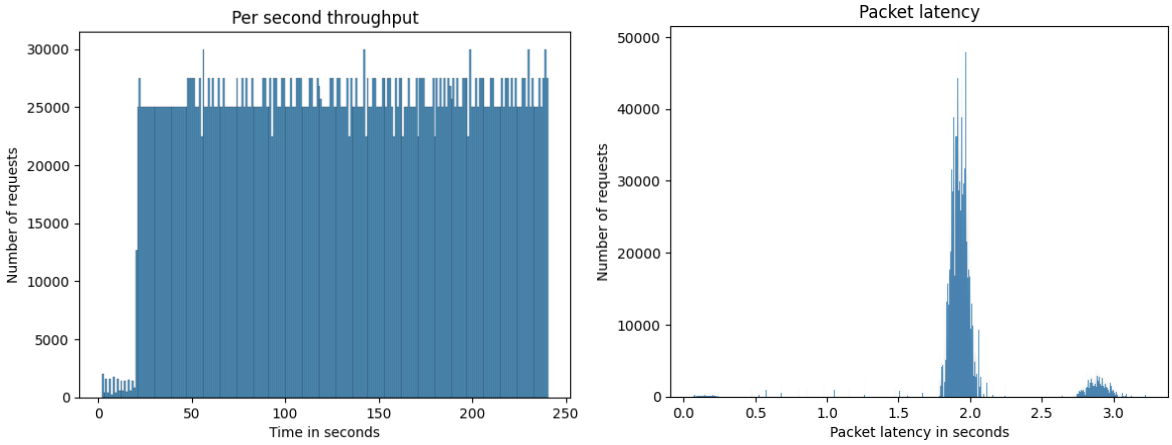
5.2.2 Burst Arrival Rate



(a) Throughput per second

(b) Packet latency

Figure 5.3: Baseline performance with burst arrival rate, setup with 8 machines



(a) Throughput per second

(b) Packet latency

Figure 5.4: Baseline performance with burst arrival rate, setup with 20 machines

Once again, we experimented over two configurations of our system, one with 8 machines and the other with 20 machines. Client requests were modeled to follow a Poisson distribution within the initial 20 seconds, followed by a burst period from 20 to 40 seconds. This burst was designed to mimic a situation where all offline transactions simultaneously come online. No additional requests were sent from the clients after this period. A particularly interesting aspect of these tests was observing the time required for all packets to be received and processed.

In the system described, the execution layer operates on a separate thread from the consensus layer, with a multichannel approach facilitating the transfer of information between these layers.

During the burst setup, the channel can become saturated due to the time taken for execution. This saturation causes the entire system to slow down, as the channel's capacity to handle incoming data is exceeded.

In the 8-machine configuration, we observed a median latency of approximately 534,342 microseconds, with the 90th percentile latency reaching around 578,803 microseconds (Figure 5.3b). The average throughput achieved in this setup was 32,000 packets per second (Figure 5.3a). Notably, after 175 clients had sent around $1e6$ packets all for which they received responses.

Expanding the scale to 20 machines, we noticed a significant increase in latency. The median latency rose to roughly 1,931,156 microseconds, and the 90th percentile latency escalated to about 2,032,760 microseconds (Figure 5.4b). In this larger configuration, the average throughput decreased to 24,000 packets per second (Figure 5.3a). This reduction in throughput, along with the increased latency, highlights the challenges in scaling. In this 20-machine setup, it took 240 seconds for each client to sent around 600,000 packets and receive the corresponding responses.

5.3 Performance with an Execution Layer

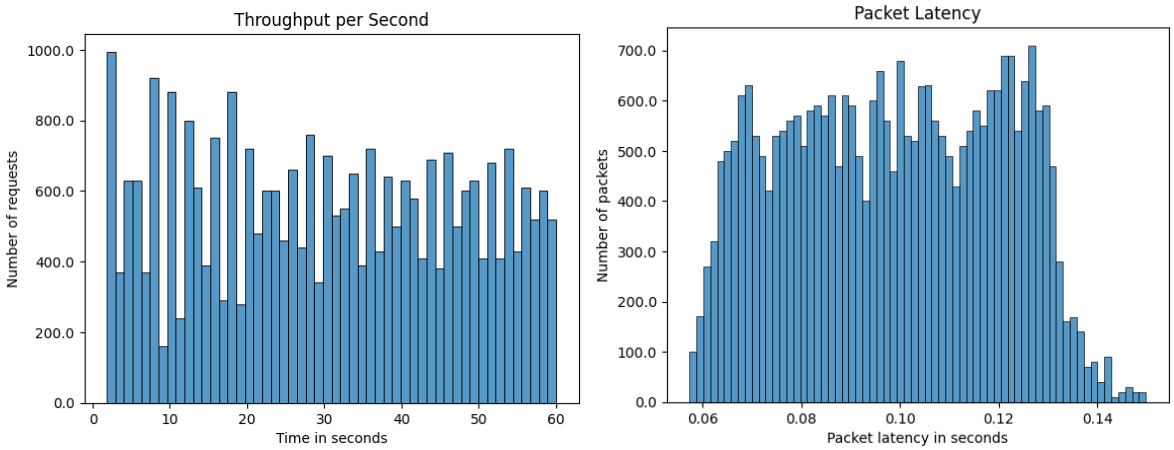
In this segment of our analysis, we evaluate the system performance within the HAT transaction framework. This consequently means that the client requests are transaction that have to be processed in the execution layer and have a payload size of 16 bytes after serialization. Hence, we can evaluate the effect that doubling the requests' size and adding an execution layer on system performance.

5.3.1 Fixed Poisson Arrival Rate

The tests conducted used two different configurations, the first involved a total of 10 machines, 5 clients and 5 replicas. The second configuration expanded the setup to 10 clients and 10 replicas. A comparative analysis of these two configurations yielded observations consistent with the baseline performance metrics we previously established.

One notable finding was that the average throughput in both configurations aligned closely with the baseline performance. Specifically, we observed a throughput of 500 requests per second in the configuration with 5 clients (Figure 5.5a), and 990 requests per second with 10 clients (Figure 5.6a). The consistency in throughput despite the increase in payload can be attributed to the still relatively small size of client requests and the Poisson arrival rate, which, in our setup, resulted in a comparatively low throughput.

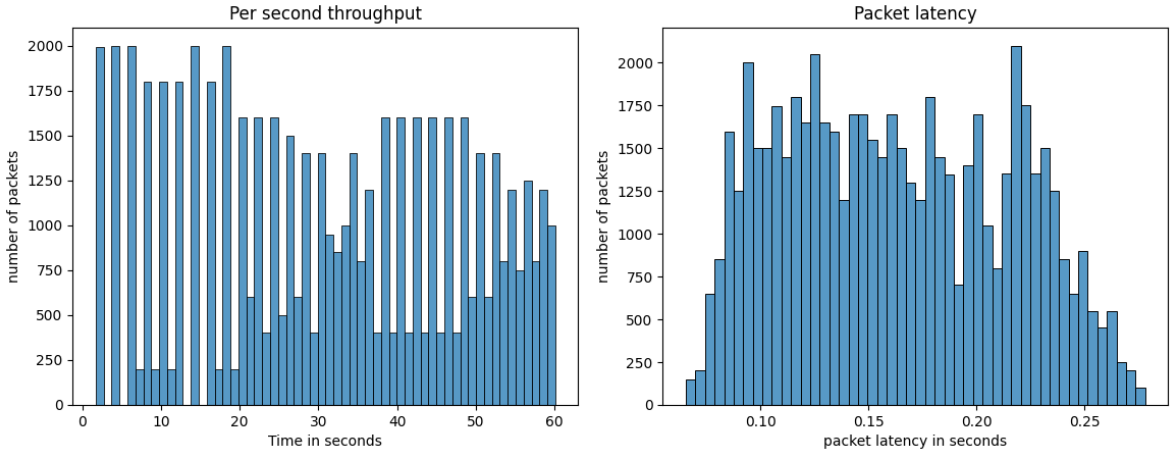
We can observe an increase in latency correlating with the increase in the number of machines. In the configuration with 10 machines, the median latency recorded was 98,704 microseconds



(a) Throughput per second

(b) Packet latency

Figure 5.5: Performance with transaction architecture and fixed Poisson arrival rate, setup with 10 machines

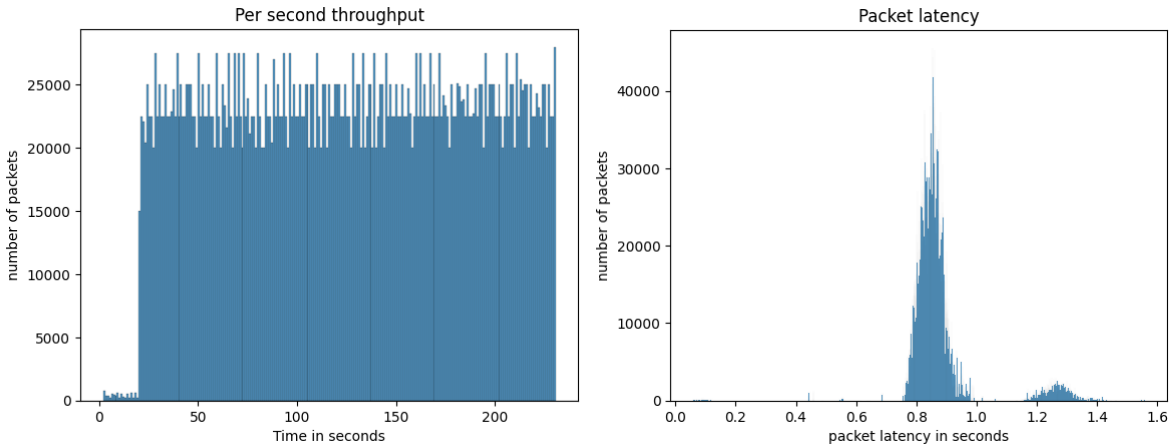


(a) Throughput per second

(b) Packet latency

Figure 5.6: Performance with transaction architecture and fixed Poisson arrival rate, setup with 20 machines

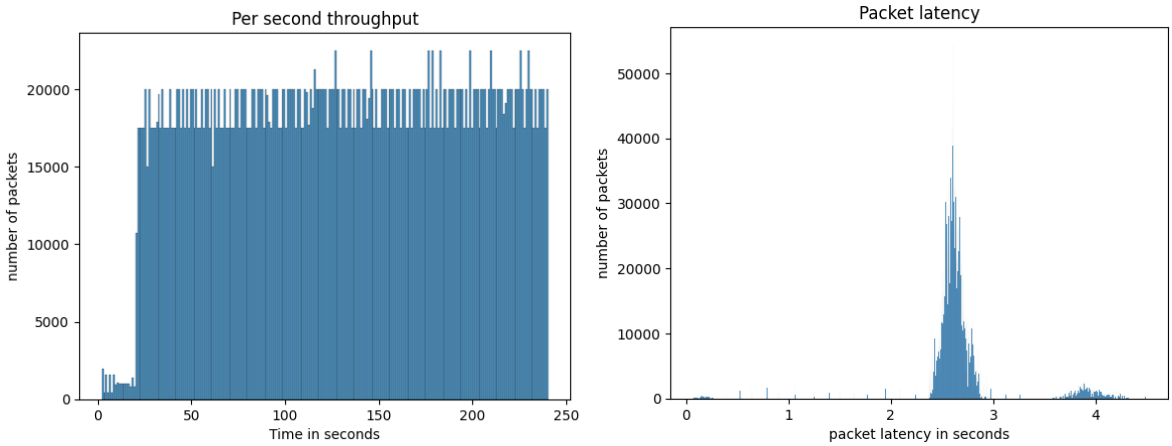
(Figure 5.5b). This latency increased in the 20-machine setup, rising to 158,892 microseconds (Figure 5.6b). Furthermore, the 90th percentile latency also demonstrated an increase, the value almost doubling going from 127,093 microseconds for the 10-machine configuration, to 233,350 microseconds in the 20-machine configuration.



(a) Throughput per second

(b) Packet latency

Figure 5.7: Performance with transaction architecture and burst arrival rate, setup with 8 machines



(a) Throughput per second

(b) Packet latency

Figure 5.8: Performance with transaction architecture and burst arrival rate, setup with 20 machines

5.3.2 Burst Arrival Rate

Once again, we evaluated our system's performance using two different configurations: one involving 8 machines and the other 20 machines. Similarly to the evaluation of the baseline burst performance, the client requests in these tests were set to follow a Poisson distribution for the first 20 seconds, transitioning into a burst period from 20 to 40 seconds. No requests were sent from the client side after the 40-second mark. A key focus of these tests was to observe and analyze the time required for the complete receipt and processing of all packets in these distinct setups as the channel becomes saturated due to the time taken for execution.

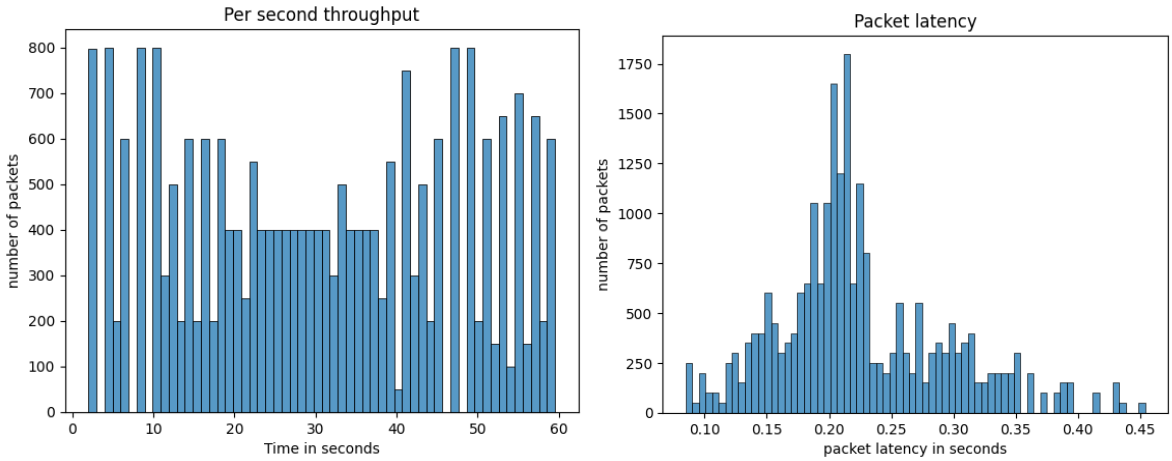
For the 8-machine configuration, the results indicated a median latency of approximately 854,031 microseconds and a 90th percentile latency close to 923,127 microseconds (Figure 5.7b). The average throughput observed in this scenario was around 22,000 requests (Figure 5.7a). We can see that it takes around 230 seconds for the clients to receive responses to all of their 960,000 requests.

Upon increasing the setup to include 20 machines, there was a noticeable change in the system's performance metrics. The median latency rose significantly to about 2,617,540 microseconds, and the 90th percentile latency increased to around 2,810,763 microseconds (Figure 5.8b). Concurrently, there was a decrease in the average throughput, which stood at 17,500 requests (Figure 5.8a). Once again, the increased latencies and altered throughput rates in the larger configuration highlight scalability challenges. We can see that it takes around 240 seconds for the clients to receive responses to all of their 430,000 requests.

5.4 Performance With System Cryptography

In this section, we evaluate the system performance with the HAT transaction cryptography framework in place. This consequently means that the client requests payload size becomes 287 bytes, hence, we can evaluate the system performance with a significantly higher client request payload.

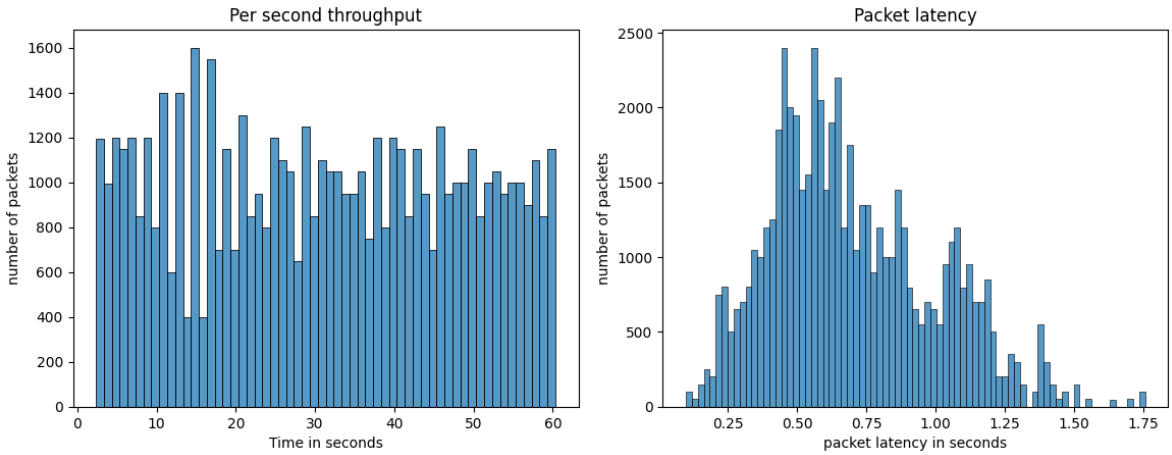
5.4.1 Fixed Poisson arrival rate



(a) Throughput per second

(b) Packet latency

Figure 5.9: Performance with system cryptography and fixed Poisson arrival rate, setup with 8 machines



(a) Throughput per second

(b) Packet latency

Figure 5.10: Performance with system cryptography and fixed Poisson arrival rate, setup with 20 machines

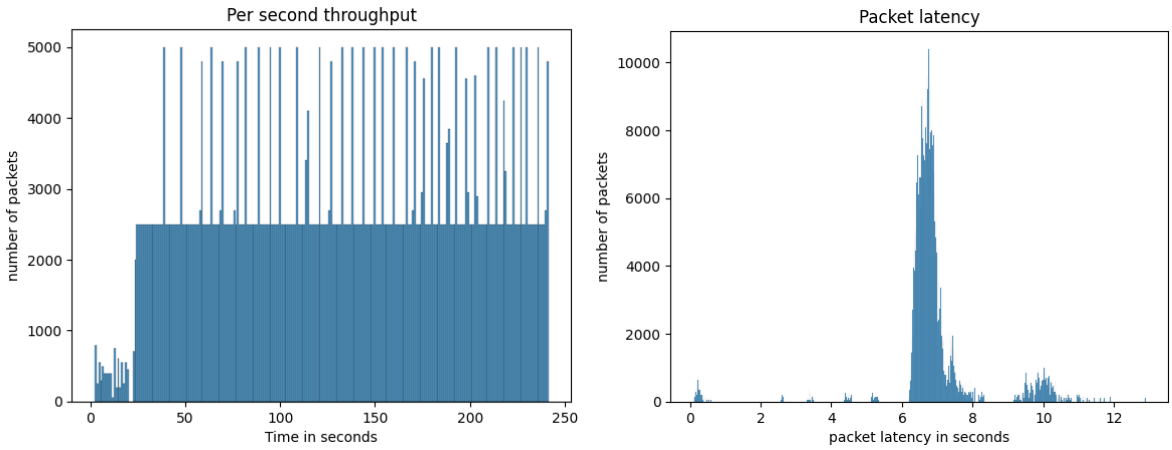
In our analysis of performance with transaction cryptography incorporated in the system, our experiments, over the two different setups, involved client requests following a Poisson arrival rate. This setup led to a relatively low throughput, which, as we will see, does not reflect the system's maximum capacity. The true capability of the system is more accurately represented in scenarios with burst traffic.

For the initial configuration using 8 machines, the system exhibited a median latency of approximately 212,025 microseconds and a 90th percentile latency of around 320,775 microseconds (Figure 5.9b). The average throughput in this setup was 393.27 requests per second (Figure 5.9a). These metrics provided a baseline for understanding the system's performance under a controlled, moderate load.

Upon expanding the system to include 20 machines, there was a notable change in performance. The median latency almost tripled to about 641,363 microseconds, with the 90th percentile latency reaching approximately 1,126,601 microseconds (Figure 5.10b). This increase in latency was a consequence of scaling up the number of machines. Interestingly, the average throughput also increased to 976.5 requests per second (Figure 5.10a). This higher throughput is attributed to the greater number of machines contributing to more requests being processed on the network.

5.4.2 Burst arrival rate

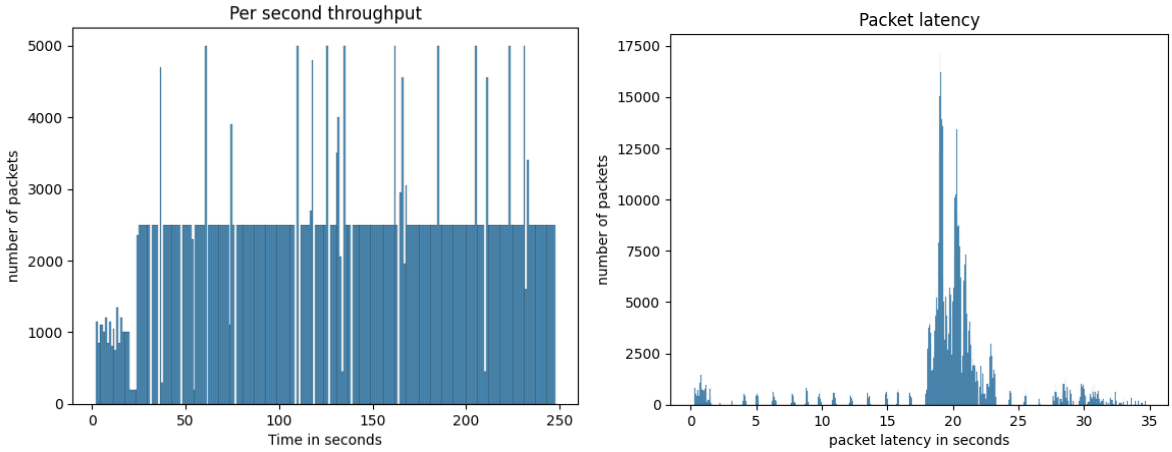
We continue to focus on two key metrics: latency and throughput, under two different machine configurations. Once again, the client requests in these tests were set to follow a Poisson distribution for the first 20 seconds, transitioning into a burst period from 20 to 40 seconds. No requests were



(a) Throughput per second

(b) Packet latency

Figure 5.11: Performance with system cryptography and burst arrival rate, setup with 8 machines



(a) Throughput per second

(b) Packet latency

Figure 5.12: Performance with system cryptography and burst arrival rate, setup with 20 machines

sent from the client side after the 40-second mark. We maintain that in the burst setting the time in takes for all client requests to be processed is an additional interesting metric as the channel becomes saturated due to the time taken for execution.

For the configuration with 8 machines, the median latency was recorded at approximately 6,751,489 microseconds, and the 90th percentile latency was about 7,719,136 microseconds (Figure 5.11b). Despite these high latency values, the system maintained an average throughput of 2,600 requests per second (Figure 5.11a). In terms of transaction handling, after 240 seconds, clients received responses for 97 % of the 180,000 requests they sent.

Expanding the system to 20 machines, we observed a significant increase in latency. The median latency rose to roughly 19,950,624 microseconds, with the 90th percentile latency reaching approximately 22,913,878 microseconds (Figure 5.12b). Notably, the average throughput remained almost consistent at 2,300 requests per second (Figure 5.12a). However, in 250 seconds, clients received responses for 91 % of their 60,000 requests.

From these observations, it appears that the throughput of 2,700 requests per second, with a request payload size of 287 bytes, acts as a bottleneck for the system. This bottleneck is consistent across both configurations and is a crucial factor in the increased latency, especially in the larger 20-machine setup.

5.5 Performance Evaluation Discussion

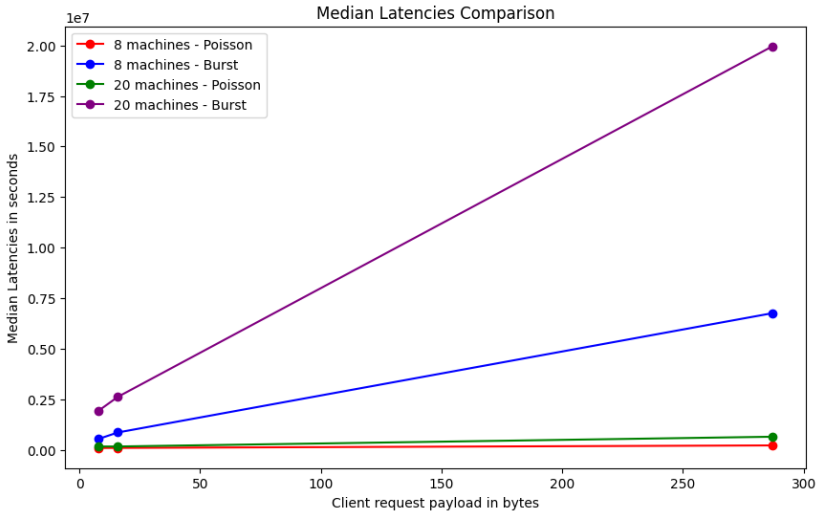


Figure 5.13: Median Latencies Comparison

The performance analysis of the HAT network using the HotStuff consensus algorithm provides critical insights into the system's scalability and efficiency, as illustrated in the provided graphs for setups with 8 and 20 machines (Figures 5.13 and 5.14). From these graphs, it is evident that as the number of nodes in the network increases, the system's throughput tends to diminish, while latency correspondingly increases.

The latency graphs show a pronounced increase when transaction cryptography is introduced, suggesting that the cryptographic operations are a substantial contributor to the delay in transaction processing. This is particularly noteworthy as it points to a bottleneck within the performance of cryptographic transactions.

In real-world applications, understanding how this scales with an even larger network is essential.

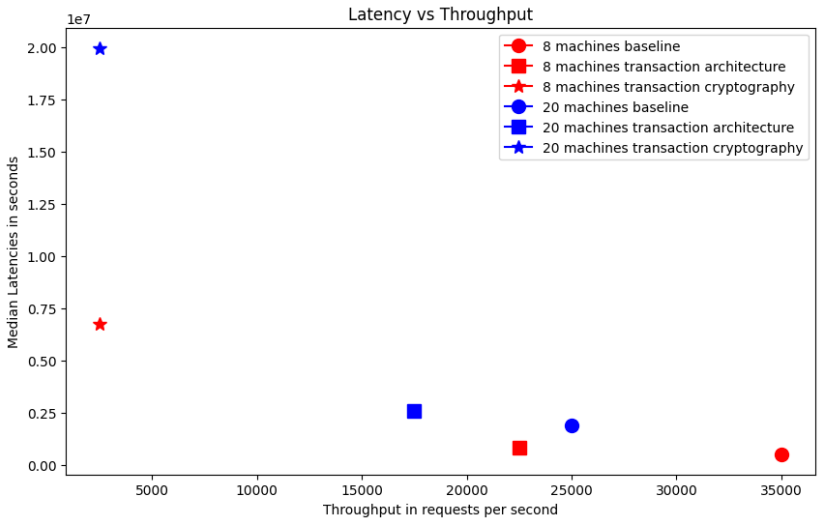


Figure 5.14: Latency vs Throughput in a Burst Setting

Given the current trends observed in the graphs, one can infer that with more machines, the system might encounter scalability issues.

Benchmarking against established systems such as Visa, Bitcoin, and Ethereum offers a perspective on the HAT network's performance. Bitcoin, with its capacity of handling approximately 7 transactions per second, is significantly outpaced by our system, which offers a marked improvement in throughput. Ethereum, which currently processes around 30 transactions per second, is also projected to leap forward with the upcoming release of Ethereum 2.0, enabling it to process anywhere from 20,000 to 100,000 transactions per second. Visa, on the other hand, typically processes about 1,700 transactions per second in reality, even though they claim a capacity of 24,000 transactions per second. This insight into the transactional throughput of these major systems underscores the competitive standing of our HAT network and the potential for it to fulfill the transactional demands within the humanitarian sector.

In conclusion, the HAT network exhibits promise in terms of throughput when compared to other blockchain technologies but faces challenges in latency, particularly when cryptographic processes are involved. To enhance its practical viability, efforts must be directed towards scaling to accommodate a larger number of nodes without compromising on speed and efficiency. Addressing these challenges is crucial for the HAT network to be a feasible alternative in the domain of digital transactions.

In the specific context of our humanitarian application, scalability and security are crucial factors. On the other hand, while speed is generally a significant concern in transaction processing, our system's support for offline transactions introduces a degree of flexibility. The immediacy of transaction confirmation is not as critical as it would be in a conventional commercial setting.

Therefore, the allowance for more processing time for client requests does not impede the ability of beneficiaries to initiate new transactions.

Chapter 6

Conclusion

This thesis explores the transformative potential of digital currencies in humanitarian aid, proposing the HAT network as a novel alternative to traditional voucher systems. HAT grants beneficiaries with greater autonomy in aid utilization, while prioritizing their privacy.

A key feature of HAT is its commitment to everlasting privacy. This is achieved through the trade-off of requiring in-person enrollment and transactions, although future iterations may include remote options for previous interactants. By guaranteeing everlasting privacy financial records remain confidential, similarly to the privacy offered by physical cash. The network's ability to support offline transactions also mirrors the flexibility and confidentiality of cash, while simultaneously offering greater transparency and accountability guarantees.

Performance analysis indicates that HAT maintains a competitive throughput rate, albeit with increased latency in larger network configurations. Considering the nature of HAT's operational context, latency might not be critically detrimental, as offline transactions can be processed later, akin to the network operating in an offline mode. This raises important considerations about scalability and the practical impact of latency in humanitarian contexts, particularly in scenarios involving offline transactions.

Further work could explore the scalability of HAT in larger networks and compare its cryptographic scheme efficiency against traditional asymmetric schemes. Future research directions include benchmarking HAT against protocols like Raft-Forensics and other advanced algorithms developed for Central Bank Digital Currencies (CBDCs), to further refine its applicability in humanitarian contexts.

The HAT network, with its innovative approach, stands as a testament to the potential synergy between digital currency technology and humanitarian initiatives, paving the way for more efficient, secure, and user-centric aid distribution models.

Bibliography

[1] Jem Bendell, Matthew Slater, and Will Ruddick. “Re-imagining Money to Broaden the Future of Development Finance”. In: ().

[2] Leander Bindewald. “Inconsistent Definitions of Money and Currency in Financial Legislation as a Threat to Innovation and Sustainability”. In: *Journal of Risk and Financial Management* 14.2 (Feb. 2021). Number: 2 Publisher: Multidisciplinary Digital Publishing Institute, p. 55. ISSN: 1911-8074. DOI: 10.3390/jrfm14020055. URL: https://www.mdpi.com/1911-8074/14/2/55 (visited on 01/12/2024).

[3] Leander Bindewald and Susan Steed. *Money with a purpose*. New Economics Foundation. URL: https://neweconomics.org/2015/05/money-with-a-purpose (visited on 01/12/2024).

[4] *CLOC – Count Lines of Code*. URL: https://cloc.sourceforge.net/ (visited on 01/18/2024).

[5] *Experimentation*. MergeTB. URL: https://mergetb.org/docs/experimentation/ (visited on 01/18/2024).

[6] Loren Gatch. “Local Money in the United States During the Great Depression”. In: *Essays in Economic & Business History* 26 (2008), pp. 47–62. ISSN: 2376-9459. URL: https://www.ebhsoc.org/journal/index.php/ebhs/article/view/191 (visited on 01/18/2024).

[7] Mohammad M. Jalalzai, Jianyu Niu, Chen Feng, and Fangyu Gai. *Fast-HotStuff: A Fast and Resilient HotStuff Protocol*. Nov. 3, 2022. arXiv: 2010.11454[cs]. URL: http://arxiv.org/abs/2010.11454 (visited on 10/05/2023).

[8] Rebecca Mqamelo. “Community Currencies as Crisis Response: Results From a Randomized Control Trial in Kenya”. In: *Frontiers in Blockchain* 4 (2022). ISSN: 2624-7852. URL: https://www.frontiersin.org/articles/10.3389/fbloc.2021.739751 (visited on 01/11/2024).

[9] Esther Oliver Sanz. “Community currency (CCs) in Spain: An empirical study of their social effects”. In: *Ecological Economics* 121 (Jan. 1, 2016), pp. 20–27. ISSN: 0921-8009. DOI: 10.1016/j.ecolecon.2015.11.008. URL: https://www.sciencedirect.com/science/article/pii/S0921800915004401 (visited on 01/12/2024).

[10] Tim Roughgarden. “Foundations of Blockchains Lectures #6: The Partially Synchronous Model, 33%, and the CAP Principle”. In: ().

[11] Bianca Schroeder, Adam Wierman, and Mor Harchol-Balter. “Open versus closed: a cautionary tale”. In: *Proceedings of the 3rd conference on Networked Systems Design & Implementation - Volume 3*. NSDI’06. USA: USENIX Association, May 8, 2006, p. 18. (Visited on 01/18/2024).

[12] Anna Stephens. “Re-imagining Money to Broaden the Future of Development Finance”. In: ().

[13] Weizhao Tang, Peiyao Sheng, Pronoy Roy, Xuechao Wang, Giulia Fanti, and Pramod Viswanath. *Raft-Forensics: High Performance CFT Consensus with Accountability for Byzantine Faults*. Nov. 2, 2023. DOI: 10.48550/arXiv.2305.09123. arXiv: 2305.09123[cs]. URL: http://arxiv.org/abs/2305.09123 (visited on 01/18/2024).

[14] Pasindu Tennage, Cristina Basescu, Eleftherios Kokoris Kogias, Ewa Syta, Philipp Jovanovic, and Bryan Ford. *Baxos: Backing off for Robust and Efficient Consensus*. Apr. 22, 2022. arXiv: 2204.10934[cs]. URL: http://arxiv.org/abs/2204.10934 (visited on 01/18/2024).

[15] Pasindu Tennage, Antoine Desjardins, and Eleftherios Kokoris Kogias. *Mandator and Sporades: Robust Wide-Area Consensus with Efficient Request Dissemination*. Sept. 16, 2022. arXiv: 2209.06152[cs]. URL: http://arxiv.org/abs/2209.06152 (visited on 01/08/2024).

[16] *The Go Programming Language | IEEE Journals & Magazine | IEEE Xplore*. URL: https://ieeexplore.ieee.org/document/6898707 (visited on 01/18/2024).

[17] *Transmission Control Protocol rfc: 793*. URL: https://www.ietf.org/rfc/rfc793.txt (visited on 01/18/2024).

[18] Learn With Whiteboard. *All Major Blockchain Consensus Algorithms Explained*. Medium. May 7, 2023. URL: https://medium.com/@learnwithwhiteboard_digest/all-major-blockchain-consensus-algorithms-explained-6934b4f5d47a (visited on 01/13/2024).

[19] Maofan Yin, Dahlia Malkhi, Michael K. Reiter, Guy Golan Gueta, and Ittai Abraham. *HotStuff: BFT Consensus in the Lens of Blockchain*. July 23, 2019. arXiv: 1803.05069[cs]. URL: http://arxiv.org/abs/1803.05069 (visited on 10/05/2023).