Alma Mater Studiorum
Università di Bologna
Campus di Cesena

DIPARTIMENTO DI INFORMATICA – SCIENZA E INGEGNERIA
Corso di Laurea Magistrale in Ingegneria e Scienze Informatiche

ANONSHARD: A PEER-TO-PEER NETWORK FOR ANONYMOUS AND DECENTRALIZED COMPUTATION

Elaborato in:
CYBERSECURITY

Relatore:
D'Angelo Gabriele

Presentata da:
Paganelli Alberto

Sessione IV
Anno Accademico 2023-2024

"Study after study has show that human behavior
changes when we know we're being watched.
Under observation, we act less free,
which means we effectively are less free."
Edward Snowden

Abstract

In today's digital era, privacy has become an increasingly rare and valuable resource. With the rapid expansion of data collection practices, individuals' personal information has become a highly sought-after resource, often traded in legal and illegal markets. Motivated by a commitment to privacy and cybersecurity, this thesis presents *AnonShard*, a decentralized peer-to-peer system prototype designed to execute computational tasks in an anonymous environment. The system aims to provide the foundation for building an anonymous computing infrastructure, expanding into a broader vision of a privacy and innovation based ecosystem. The research explores the principles of anonymity, the motivations behind anonymous computing, and the technical challenges associated with building such a system. The thesis details the system's design, architecture, implementation choices, and the trade-offs made during development due to time constraints and complexity. Additionally, evaluations and scalability tests are conducted with a secondary prototype used to analyze the Tor network's behavior. This work aims to deliver a functional and flexible prototype that can be the foundation for developing a privacy-preserving computing ecosystem. At the end of the thesis, the implementation of a reward system, the introduction of different node typologies, and other features aimed at incentivizing participation and enhancing the sustainability of the system are discussed.

Contents

List of Figures

Chapter 1

Introduction

In this section, my thesis project will be introduced, explaining the context in which it is placed, what led me to choose this topic and the objectives I wanted to achieve.
The structure of the document will be presented, to provide the reader with an overview of the document.
Since I was a child, I have always considered privacy as a fundamental right of the individual, I have always tried to protect it and to be aware of the information I share.
Today and with an eye to the future, I consider privacy an increasingly rare commodity, which is increasingly challenging to protect.
This has led me to the world of cybersecurity, combining my passion for technology and my desire to protect privacy.
In this thesis project, I tried to contribute to this huge challenge, trying to imagine an ecosystem in which anonymity and privacy are guaranteed.
In recent years, there has been a huge and ever-increasing interest in comparing data obtained from people, generating real markets, legal or illegal, in which people's preferences, such as products, are sold to the highest bidder.
Not to mention the fact that personal data is increasingly the subject of theft and violations, as evidenced by the numerous cyber-attacks that have hit large companies and institutions.

With this scenario in mind, I decided to deepen this project.
Anonymous Shard consists of a decentralized peer-to-peer system to which it is possible to submit generic work units called Tasks, to execute them in a protected and anonymous environment leading to the creation of an Anony-

mous Computing system.
My goal was to create a working and easily extensible prototype that could be used as a basis for future developments and research.

A brief overview of the document structure is presented below.

1. **Chapter 2—Introduction to the concept of anonymity**: the origins of anonymity and its evolution over time. Furthermore, anonymity is a very vast and controversial theme; the value of anonymity is analyzed.
   Historical and social aspects of anonymity are analyzed, to better understand the context in which the project is placed.

2. **Chapter 3—Motivations**: Concept of anonymous computing at the base of the system. The absence of such a system is underlined.
   Analysis of the main functionalities, interactions between nodes and possible extensions to obtain a generic anonymous computing system.
   The proposed prototype and job units flows are explained with a more in-depth analysis to better understand the basic operation of the system.
   Furthermore, a second prototype created to perform measurements on the Tor network is also analyzed.

3. **Chapter 4—Design and implementation of the system**: technical details of the system and design choices are analyzed.
   There are some usage scenarios, use cases and diagrams to understand how the system has been designed.
   Within this chapter, there are also the motivations that led me to choose certain technologies and the difficulties encountered during development.
   Design choices, architectures and all that were partially explored or analyzed in detail and later discarded due to time constraints and complexity will also be examined, as it could be implemented in the future.
   Everything that has been implemented at a technical level, the methodology followed and how the system will have to be distributed and

tested, is analyzed in this chapter.

4. **Chapter 5—Testing and evaluation of the system**: the tests carried out on the system, and the metrics used to evaluate the system are analyzed.
   In particular, how the system can easily scale, what components can be improved or added considering a huge number of nodes and a real use situation.
   The measurements made with the second prototype will also be analyzed in this chapter.

5. **Conclusions**: finally, the process for creating the system, the system itself and the results obtained at a high level are analyzed, with a critical analysis towards the goal and the next possible extensions.

Chapter 2

Introduction to Anonymity

2.1 Definition and Evolution of Anonymity

Anonymity is a fundamental concept in human interactions and communications. Different individuals may define anonymity in various ways, evoking different emotions and interpretations. The definition of this term can be broad, ranging from the absence of personal identification to the concealment of specific traits or actions. Different visions of anonymity have evolved over time, reflecting changes in technology, society, and human values.

Without focussing directly on the technical aspect, the concept of anonymity has deep roots in human history, philosophy and psychology. A relevant aspect is the contextual nature of anonymity. Nissenbaum emphasizes that anonymity is context-dependent [15], meaning an individual may be anonymous in one specific environment but identifiable in another. This highlights the complexity of the topic, as it is not an absolute state but a relative one, influenced by different factors.

A definition that I found particularly interesting is the one by Wallace, who describes anonymity as *the non-coordinatability of traits in a given respect* [11]. This means an individual is anonymous when certain characteristics or actions cannot be linked to identifiable traits such as name, location, or social identity, introducing also the difference between anonymity and identifiability.

This definition, declined in this digital age, where online interactions and data collection pose significant challenges to anonymity, adhere perfectly. Furthermore, differentiating between anonymity and identifiability is essential, as the latter impact directly on anonymity and privacy. Anonymity,

therefore, is not only about being nameless or hidden; it involves preventing the coordination of distinct features that could lead to identification. This, combined with technological advancements, social norms, and ethical considerations, shapes the evolving concept of anonymity. Adapting the definition to a technological prospective, anonymity can be seen as the ability to conceal one's identity in digital environments. Digital footprints, including IP addresses, cookies, metadata, and other tracking technologies, pose significant challenges to this concept.

Returning to the contextual nature of anonymity, it is essential to consider the multiple layers of identifiability in digital interactions. For instance, someone using a pseudonym in an online forum may remain anonymous within the platform (first context) but could be identified through other digital traces by service providers or authorities (other context). But the same logic applies to social structures and behavioral patterns, which can be used to identify individuals even in a non-technical context.

In this case, and particularly in the digital declination, the definition of anonymity is therefore more complex and multifaceted. It involves multiple degrees of identifiability, de-anonymization risks, and the interplay between privacy and security.

For the concept of anonymity to be meaningful, it must be considered in a broader context, encompassing technological, social, and ethical dimensions. This nature of anonymity raises ethical and legal considerations. While the ability to identify individuals is necessary in cases of harmful actions, the exploitation of personal data for surveillance, censorship, and behavioral prediction by governments and agencies remains a significant concern.

In this context, the issue revolves around the misuse of personal data for unethical purposes, going beyond the mere technical aspect of anonymity to the privacy and security implications. However, by shifting the perspective from the individual level to a broader market-driven approach, aligning more with Andrew Grove's vision [28], it becomes even cleaner how vast and delicate this issue truly is and how "defensive paranoia" can be beneficial.

The power of data aggregation can lead to discrimination, manipulation, and large-scale privacy violations. Furthermore, information can be weaponized to influence public opinion, as demonstrated in the Cambridge Analytica scandal [23]. However, the risk is to mesh the concept of anonymity with the one of privacy. Privacy is about control over personal information, while anonymity is about concealing identity. Anonymity, therefore, is a

tool for preserving privacy, enabling individuals to communicate and express themselves without fear of persecution.

The United Nations reaffirms privacy as a fundamental human right, emphasizing the abuses as highly intrusive acts and the importance of protecting individuals from unwarranted and unlawful surveillance permitting to express themselves freely both offline and online [29]. Also, at an American Association for the Advancement of Science (AAAS) conference, experts acknowledged anonymous communication as a fundamental human right, affirming that individuals and organizations should have the discretion to determine their level of anonymity [17].

In particular, I agree to define it a right, since its vital role in enabling free expression, protecting activists, researchers, and individuals living under oppressive regimes. It must safeguard privacy in sensitive discussions and shield people from discrimination, repression, and surveillance. Moreover, as a tool, it can ensure that no data is made available to third parties, protecting against data aggregation and profiling. However, as mentioned, it presents challenges in accountability, as bad actors may exploit anonymity to evade consequences. More on this aspect can be found in the Controversies Section 2.4.

The evolution of anonymity reflects the tension between privacy, security, and accountability resulting in more complex and challenging scenarios. The science of cryptography and modern techniques plays a crucial role in preserving anonymity. At the same time, tension between anonymity and regulation persists, with governments and corporations seeking to limit anonymity for security purposes, while excessive restrictions risk undermining civil liberties.

End-to-End Encryption (E2EE) is a quite actual example of this conflict. Governments often view E2EE as a security risk, as it prevents access to communication contents even for law enforcement. In November 2020, the European Council proposed a resolution addressing the use of E2EE in messaging applications, highlighting the challenge of balancing privacy protection with investigative needs [16].

Yet another example from just these days related to E2EE is the Apple and the United Kingdom government dispute. Apple disabled Advanced Data Protection (ADP: feature that encrypts iCloud backups) in the UK after government pressure to create a global backdoor for law enforcement, grounding the request on the *Online Safety Act*. Apple refused and instead disabled ADP for the UK users, making users' backups potentially available

upon legal request. This choice to disable the feature instead of creating a backdoor might not deter other nations. Governments could push tech companies to comply, making privacy compromises in exchange for avoiding legal conflicts. Beyond the ambiguity of the request itself, it set a precedent and highlighted also how tech companies are becoming increasingly strategic players in the privacy and security landscape.

The resolution reflects the balance between protecting citizens' fundamental rights and enabling authorities to investigate criminal activities in case of need.

2.2 The Value of Anonymity

As introduced in the Definition Section, anonymity is a multifaceted concept that plays a crucial role in preserving freedom of expression and privacy. It enables individuals to voice opinions without fear of retaliation, serving as a shield for whistleblowers, journalists, and activists [21]. This value on fostering privacy and free expression is undeniable. It allows individuals to engage in sensitive discussions without the risk of discrimination or repression.

The Cypherpunk movement, born in the early 1990s, recognized this long before the actual digital age. Rooted in the belief that privacy is a fundamental right, cypherpunks advocate for the use of cryptography for a free and open society. As David Chaum discussed, cryptographic tools enable individuals to participate in digital interactions without exposing their identities, preventing surveillance and censorship [24]. Anonymity is not about hiding wrongdoing; it is about preserving freedom. In a world where governments and corporations track, analyze, and monetize every online action, anonymity safeguards individuals from oppression, discrimination, and unwarranted scrutiny. May, described how cryptographic anonymity could create a world where individuals could interact without centralized control, ensuring freedom from coercion and oppression [12, 22].

Anonymity can bring to true autonomy. Without it, individuals are pressured to conform to societal expectations, limiting creativity, innovation, and dissenting opinions. How also Christopherson highlighted, anonymity can favor also mental well-being, offering spaces where individuals can express themselves freely and with no fear of judgment [20]. Assange also highlighted how cryptography is not just a tool for privacy but a weapon

against authoritarianism, ensuring individuals retain power over their own information [25].

This prospective may seem idealistic, but it is a vision that has inspired many and continues to drive the development of Privacy-Enhancing Technologies (PETs). Another fundamental work is *Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms* by Chaum, who introduced the concept of Mix Networks (or MixNets), enabling anonymous communication by shuffling and re-encrypting messages [13]. This laid the foundation for modern technologies like Tor, anonymous remailers and others. However, anonymity is constantly under attack. Governments impose regulations, social platforms push real-name policies, and financial systems attempt to erode private transactions. The Cypherpunk movement reminds us that without proactive resistance, privacy will be eroded. True digital freedom lies in our ability to communicate, transact, and express ourselves without external interference, and it is not an option for an open and liberated society.

2.3 Social Considerations

Anonymity is also inherently tied to social contexts. This section explores its social implications, trying also to analyze the actual social attitude towards anonymity and privacy preservation. In this way, it is possible to better understand the context in which the project is placed. An emphasis is on the need to reintroduce anonymity-preserving tools into everyday life to safeguard privacy and freedom of expression. Additionally, the discussion extends to the inherently non-anonymous nature of social networks and digital platforms, highlighting how these systems shape users' online identities giving rise to privacy concerns. Finally, the importance of education on privacy and data protection is considered essential for fostering a culture of privacy awareness.

Maintaining a high level of privacy through anonymity tools is crucial in this digital age but requires a conscious effort. Even individuals, who are often labeled as "paranoid" or accused of "overreacting" in their concerns about privacy sometimes, need to choose to trade privacy for convenience, sharing personal information on digital platforms or social media. A key point here is that privacy, like security, is frequently a "trade-off": these privacy-conscious individuals make an active choice to share their data to use a service or platform. But the risk is that sometimes, only the usage

of certain technologies can lead to unintended data exposure, without the average user being aware of it.

Another fundamental issue is that many users often disclose personal information without fully considering the consequences, ingenuously. Even when not directly identifiable, digital footprints, metadata, and behavioral patterns, as mentioned in the Definition and Evolution Section 2.1, can still reveal their identity. The problem is that who is not technically prepared cannot understand if not educated. This problem is exacerbated by the modern surveillance landscape, where industries and government agencies systematically collect vast amounts of data for profiling and monitoring.

A particularly pressing concern today is therefore the widespread lack of awareness regarding privacy risks, further aggravated by the absence of proper education on the subject. This knowledge gap is especially critical in digital environments, where individuals frequently expose sensitive personal information without grasping the potential repercussions. As Solove argues in [14], the common argument "I've got nothing to hide" is misleading and oversimplified. Privacy is not merely about secrecy but about control and autonomy over one's personal information. This vision is supported also by Snowden, a former NSA employee turned whistleblower, who highlighted this sentiment in his book [27].

The rise of social media and digital platforms has further complicated anonymity. While these spaces provide opportunities for self-expression and connectivity, they also collect a lot of user data for targeted advertising and content personalization. Furthermore, they gradually introduce new requirements to access their services, continuously refining their datasets. A significant amount of information can be inferred from such data, including political affiliations, health conditions, and personal preferences, details that even those who claim to have "nothing to hide" maybe would likely prefer to keep private. The absence of anonymity in digital systems is not merely a technical issue but a deliberate choice that fuels market models based on user identifiability and preferences.

The debate on anonymity and privacy remains ongoing, reflecting the complex interplay between individual rights, societal norms, and technological advancements. The development of privacy-enhancing technologies, such as Tor or Signal, represents a step toward mitigating these concerns. However, these tools are often insufficient on their own and must be used with caution and awareness of their limitations, otherwise the problem should be

slightly mitigated but not solved. In particular, the risk is that the usage of these tools can lead to a false sense of security, as users may not fully understand the risks and limitations of these technologies, reinforcing the concerns raised by Ohm [26]. This problem is exacerbated by the lack of awareness and education on these issues, not just in terms of privacy but cybersecurity as a whole. Users often also lack the knowledge of accessible tools that can help protect their privacy and anonymity, or if they do, they may not understand how to use them effectively. In general, there is a lack of awareness regarding the information that is shared, even through seemingly simple actions such as performing a web search at a particular time or using a Wi-Fi connection rather than a mobile one. I believe that education on these issues would constitute a significant step forward in addressing this gap.

2.4 Controversies

After establishing that anonymity is a fundamental right and a tool for preserving privacy and freedom of expression, it is essential to address the controversies and present ethical dilemmas. Studies suggest that anonymity can encourage antisocial behaviors [18], and the main concern is that simultaneously with the benefits, the missing accountability potentially shields bad actors from repercussions. In this case, therefore leading to concerns about public safety and law enforcement's ability to monitor and prevent harmful actions.

As introduced in previous sections, also regulamentation and oversight are controversial aspects. Specifically, it's still a problem of accountability for actions taken behind anonymous identities and in particular for the potential misuse of anonymity for malicious purposes. Suler, in his work, highlighted the phenomenon of online disinhibition, concluding identifying both beneficial and toxic aspects of disinhibition [19].

The balance between privacy, security, and accountability is delicate, and the misuse of anonymity can have severe consequences, it's clear. The point here is that the benefits of anonymity must be weighed against the risks, and moreover, it's not new that technology, in general, can be used for both good and bad purposes.

Thinking about the possible power imbalances that can arise from the erosion of anonymity or the ethical dilemmas that can emerge from its protection is essential, and the societal impact of anonymity remains a point of

contention. This duality fuels ongoing debates about anonymity, and as mentioned, this trade-off is not new; similar debates have emerged throughout history with nearly all transformative technologies. In my view, anonymity is a powerful tool that must be protected and preserved. While it should be continually assessed for its ethical and social implications, it must also be maintained in a balanced system.

2.5 Conclusion

The concept of anonymity as understood is multifaceted, evolving alongside technological and societal changes. While anonymity enables freedom of expression and protects individuals in various contexts, it also raises ethical and legal challenges.

In the following chapters, one prototype of anonymous tool inspired more to the Cypherpunk Movement vision will be presented, focusing on the value of anonymity. This thesis emphasizes therefore the anonymity as a right and the importance of preserving it in this current context. The need of a stable ecosystem that can be used and supported by everyone, without the need of a central authority overseeing it, which is a significant strength, will be highlighted. The usage of innovative technologies and the usage of privacy-enhancing tools will be discussed, trying to keep the research on this topic alive.

Chapter 3

Motivations

In this chapter, the concept of anonymous computing at the base of the system is analyzed. The absence of such a system is underlined and the need for implementation is discussed. The relevance of anonymous computing, the proposed prototypes and various data flows are explained with a more in-depth analysis.

3.1 Anonymous Computing

Anonymous computing refers to a paradigm that prioritizes privacy, anonymity, and resistance to surveillance or censorship. It enables users to perform computational tasks or job units without revealing their identity, location, or data to untrusted or unwanted parties.

The foundation of anonymous computing is built on cryptographic protocols, peer-to-peer networks, and privacy-enhancing technologies (PETs), such as onion routing. As privacy becomes increasingly rare in today's world, the demand for systems that prioritize anonymity and security continues to grow. It has become common to promote software by highlighting its security, lack of data collection, or privacy-preserving features. In these last years, a key realization has emerged: so-called "free" software often treats users as the product: a practice that raises serious concerns.

By following this paradigm and increasing decentralization, a peer-to-peer ecosystem can be created, enabling unrestricted computation without the risk of surveillance, censorship, or external control. This, also, minimizes risks associated with data breaches ensuring that no single entity has absolute

control over job details information and computational resources. While such a system may sound utopian or even intimidating, its necessity is becoming increasingly clear in today's world.

This approach is particularly crucial in environments where privacy is essential. While some societies may take privacy for granted, there are regions where government overreach and censorship pose serious threats to personal freedom. For those who believe such systems are unnecessary today, it's important to remember that the need for privacy and anonymity can arise unexpectedly. Preparing in advance may seem paranoid, but as Andrew Grove said, "Only the paranoid survive". While Grove's statement was originally aimed at businesses and strategic market changes, the principle can be applied here. Vigilance against threats to individual freedom is equally necessary. In a world where the balance between personal liberty and governmental control is constantly shifting, expecting potential risks to privacy is not just prudent, but essential for safeguarding freedom.

Anonymous computing from this point of view can be a life-saving technology. Furthermore, beyond ensuring privacy, this system can foster a collaborative environment where resources are shared securely and efficiently.

By integrating decentralization and opening the system, the computational resources within it would belong to individuals or volunteers willing to share their capacity, making it fundamentally different from traditional cloud services. This ensures that, even in extreme scenarios of heavy censorship, people always have a reliable platform to operate freely without fear of surveillance.

Imagine a cloud computing service similar to AWS, but instead of relying on centralized data centers, the computing power comes from privately owned machines within an open and free ecosystem. While achieving this level of decentralization is challenging, it is becoming increasingly necessary in a world where privacy is under constant threat.

Naturally, like any powerful technology, anonymous computing could be misused for illegal activities. However, its core purpose is to empower individuals with privacy-preserving tools rather than to facilitate wrongdoing. Ethical considerations are crucial when developing such a system, but its potential to serve the common good concerns about potential misuse. The focus should be on ensuring that this technology remains a force for positive change, enabling freedom and security for those who need it most, as said also in the Introduction to the Anonymity Chapter 1.

3.1.1 Existing Solutions

In this section, some main technologies that can bring to an anonymous computing system and some existing solutions are analyzed.

Actual cloud providers offer a wide range of services, from Infrastructure as a Service (IaaS) to Software as a Service (SaaS), but they lack a fully anonymous computing environment. In certain cloud environments, users can deploy anonymous virtual machines (VMs) or containers to run workloads in isolated and private environments. These VMs are designed to prevent cloud providers from accessing or monitoring their contents. If correctly integrated, privacy-enhancing technology can obfuscate users' identities and network traffic. Some providers offer privacy-preserving solutions that allow data anonymization before cloud processing. However, they focus solely on data privacy, lacking customization options for users to control the execution environment or enhancing anonymization beyond the data one.

One particular actual solution provided by cloud services is also the use of Trusted Execution Environments (TEEs) to secure data processing. These technologies enable secure processing by isolating sensitive data in trusted enclaves in a way that neither the cloud provider nor other external parties can access it, by using hardware-based security mechanisms in this case. It is an extra layer of security that can be used to enhance privacy and confidentiality in cloud environments, but as mentioned, the communication if not properly secured can still be detected and traced.

An actual solution to ensure secure communication is the use of the Tor or I2P network that exploits encryption to anonymize users' traffic.

One older analyzed solution that exploits the Tor network is the *AnonymousCloud* system. In this application, the Tor onion routing has been exploited for customers to anonymously communicate job units and data to the system. An anonymous authentication system based on public-key cryptography facilitates billing of anonymous customers without linking their private data to their identities. In this case, it has been demonstrated that the system provides superior data ownership privacy even when a large percentage of the cloud is malicious [1]. A possible con in this system is the lack of transparency for internal cloud resources due to its architecture beyond the centralized nature. The architecture is master-slave, where the master is responsible for the management of the system, and the slaves are responsible for the computation of the tasks. Another limitation is the inability to

extend or enhance the system. As a result, it remains a standalone system rather than evolving into an ecosystem of nodes that can share resources and computation, while ensuring anonymity for users as base requirement.

One always good solution to enhance privacy if a simple OS and not an infrastructure is needed, is the use of a particular OS: Tails. It is designed to be booted as a live OS from a USB stick, and routes all internet traffic through the Tor network.

What is missing in the current state is a fully decentralized system that, exploiting all the available technologies, can guarantee the user the highest level of anonymity and can compute general tasks in a secure and private environment. Furthermore, nodes in the distributed system can leverage processors with TEE-like capabilities or exploit the one-shot feature idea, similar to Tails, providing layers of protection. This possible enhancement will be analyzed in the Future Development section 5.3.

3.2 Background

This section introduces the key technologies and components that are essential for understanding the prototype and to provide a clearer insight into the system. At the end of the Design and Implementation Chapter 4, another more in-depth analysis of the involved technologies is done. This overview aims to clarify the rationale behind the choices made and enhance the reader's overall understanding of the system. This prototype, as the main idea, is a commitment to freedom and openness, with a focus on decentralization rather than centralization, preferring technologies that empower users and promote autonomy.

3.2.1 Decentralization

An important choice taken during the analysis phase has been the decentralization of the system, influencing and posing constraint to the entire system architecture. What brings to a decentralized system is the fact that no single entity with absolute control over the system is wanted. Some benefits are:

1. **No Single Point of Failure**: A centralized system relies on a single authority or infrastructure, making it a prime target for surveillance,

censorship, or data breaches. If the central server is compromised, the anonymity and security of all tasks are compromised.

2. **Censorship Resistance**: A centralized system can be controlled, blocked, or shutdown by authorities, limiting access and making the system vulnerable to political or legal restrictions.

3. **Trust-less Architecture**: In a centralized system, users must trust the provider to act ethically and protect their privacy. However, history has shown that centralized services can be subject to data leaks, insider threats, or compliance with surveillance programs.

4. **Increased Security Against Attacks**: Centralized systems are attractive targets for hackers since a single breach can expose a large amount of data. In a decentralized system, data and processing are distributed, making it more difficult for attackers to gain access to sensitive information. A downside in this case is that all attacks on decentralized systems are possible, but no system is immune to attacks.

5. **Community-Driven and Self-Sustaining**: This decentralized network is released as open-source and possibly maintained by a community devs rather than a corporation. This ensures transparency, adaptability, and long-term sustainability without reliance on a single company's policies, funding, or business interests. In this case, the concept of decentralization is crucial for the system.

3.2.2 The Tor Network

The Onion Router (Tor) network is a decentralized protocol that uses onion routing to enable anonymous communication over the internet. Tor routes internet traffic through a series of volunteer-operated relays to obscure the user's location and usage patterns, providing anonymity and preventing traffic analysis. In the context of this system, Tor is employed to ensure that communications remain secure and private. By leveraging the Tor network, the system fosters a high degree of privacy and security, aligning with its goal of decentralization and anonymity. How Tor has been exploited and an explanation of the services offered by it can be found in the Design and Implementation Chapter 4.7.2.

3.2.3 SOCKS Proxy

To protect the privacy of users in the network, the system uses a SOCKS5 proxy. It is a general purpose proxy that sits at layer 5 of the OSI model and uses the tunneling method. The usage ensures that interactions between nodes remain confidential, particularly important in a decentralized environment, where maintaining anonymity and safeguarding communication is essential. The integration of a SOCKS5 proxy, precisely, is necessary to route traffic to the Tor network.

This type of proxy also provides greater flexibility to the system, as it allows traffic to be routed through different networks with varying protocols (I2P or VPNs). Additionally, it can be bypassed in certain cases to get a balance between performance and security when security is not a primary concern. A list of pros, how can be enabled or disabled in this system, and the snippet of code to enable it can be found in the Design and Implementation Chapter, specifically in the SOCKS5 Proxy Section 4.7.2.

3.2.4 IPFS as Storage Layer

The InterPlanetary File System (IPFS) is used as the storage solution, providing a decentralized repository for files and data. Unlike traditional centralized storage, IPFS distributes files across a network of nodes, where each file is divided into chunks and referenced by a unique cryptographic hash. It is designed to be resistant to censorship and single points of failure, making it an ideal choice for a decentralized system that values security and user autonomy. A problem derived and then discussed from the use of IPFS, not depending on the current system but for IPFS itself, is the lack of encryption for the stored data. Basically, whoever has the content-identifier can access the data. This can be a problem if the data is stored in clear because sensitive information can be exposed but can be solved by encrypting the data before storing. A deep analysis of IPFS and the tool exploited by the system to interact with it, Pinata, can be found in the Design and Implementation Chapter 4. Instead, the used technique to guarantee data confidentiality can be deepened in the Security Patterns Section 3.2.3.

3.3 Prototype

In this section, the proposed prototype and job units flows are explained. In particular, the analysis of the main functionalities, interactions between nodes and possible extensions to get a generic anonymous computing system. Only in the conclusion chapter the future work and the possible extensions will be discussed, here only some hints will be given.

The goal since the early stage of the project was to create a working and easily extensible prototype that could be used as a base for future developments and research about Anonymous Computing. The biggest constraint in this case was the time, so the focus was on the basic functionalities and the interactions between main parts to provide a proof of concept. To anonymize communications, the system exploits the Tor network between nodes while work to guarantee anonymity also for the computation itself needs to be done, but some hints are present.

As mentioned, the software is not something really mature that can be directly used in production.

The design choices focus on finding a good balance on the system design, to ensure that every subcomponent could be easily replaced or extended. But, due to the anonymity requirement, the system is not so simple to design and implement, so the focus was basically on the security beyond the design.

The implemented main abstractions are the following:

1. **Node**: a single entity in the network that can be of two different types: a **Provider** or a **Consumer**.
2. **Discovery Mechanism**: a component that allows nodes to discover each other and communicate.
3. **Task**: a generic work unit that can be submitted to the network for execution.
4. **Executor**: a component that executes tasks on a node.
5. **Evaluator**: a component that decides if a node is suitable to execute a task.

Every technical detail can be found in the Design and Implementation Chapter 4. What is important to understand here are the data flows and the main functionalities of the system.

Starting from the nodes, their interactions, the task flow analysis to the storage of the result for a generic task, the system will be analyzed from a high level point of view.

3.3.1 Nodes

To introduce the concept of nodes, it is important to understand the two main types of nodes in the system: Providers and Consumers. A node is a generic participant in the network, and a categorization in active and passive nodes in terms of network engagement has been done. While the consumer is a passive participant, the provider is active. In this sense, I would like to underline the fact that the provider knows part of the network, while the consumer might know even just one node to which submit the tasks being a more passive participant.

As understood, providers are nodes that offer computational resources to execute tasks, while Consumers are nodes that submit tasks to be executed.

Moreover, only these two types of nodes have been implemented, but the system can be easily extended to include other types of nodes, such as *Verifiers* or *Maintainers*, to enhance the system's overall capabilities. The concept of peer discovery, task submission and execution will be analyzed in the Future development Section 5.3.

3.3.2 Interactions

Once main roles for the nodes are defined, the interactions between them are analyzed. Now, the discovery mechanism used by the providers and the task submission flow for both consumers and providers will be analyzed.

Discovery Mechanism

During the discovery phase, providers need to find other nodes in the network to establish connections and offer their services. This is a crucial step in the system, as it allows nodes to communicate, collaborate and exchange resource metrics and capabilities. The discovery mechanism is responsible for facilitating this process, ensuring that nodes can find each other efficiently and securely. Moreover, it should handle potential faults. This mechanism is based on a Distributed Hash Table (DHT) and inspired by the Kademlia

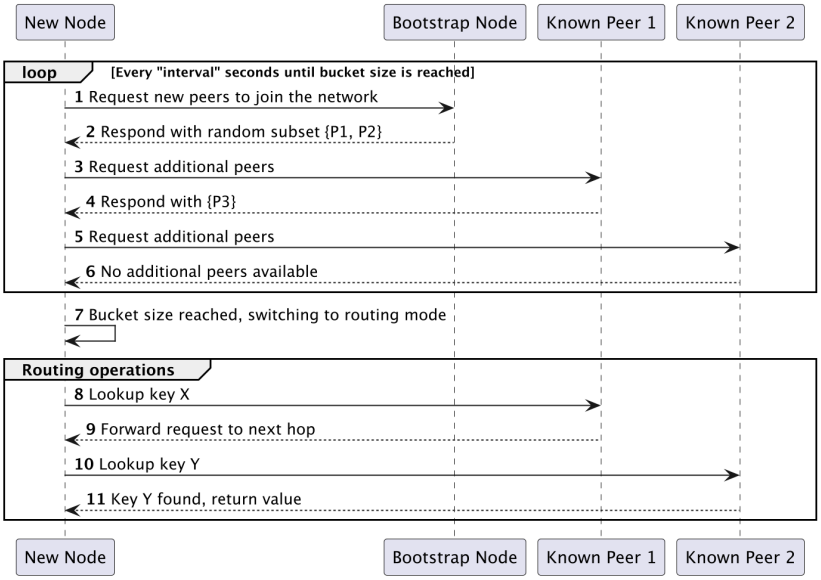

Figure 3.1: Discovery Mechanism Kademlia inspired

protocol [3]. Each node maintains a routing table that contains information about other nodes (neighbors) in the network.

By using DHTs, nodes can discover and connect to other nodes without relying on a central server or authority. Another pro is that nodes do not need to know the entire network topology, but only a subset of nodes to establish connections. This approach ensures that the system is scalable, fault-tolerant, and quite resistant in case malicious nodes have been introduced.

What is missing in this case and needs to be implemented is the establishment of a distance metric between nodes, to ensure that nodes with similar capabilities or situated nearby are connected. Following this direction, a full Kademlia protocol implementation can be done. A possible solution is to use metrics based on the latency between nodes or, since the system is based on the Tor network, to use the Tor circuit establishment time as a metric. Also, the exploitation of Entry and Exit nodes in the Tor network can be used 4.7.2. A lot of work needs to be done in this area, but it is a good starting point for future developments. More on this can be found in the Future Development Section 5.3.

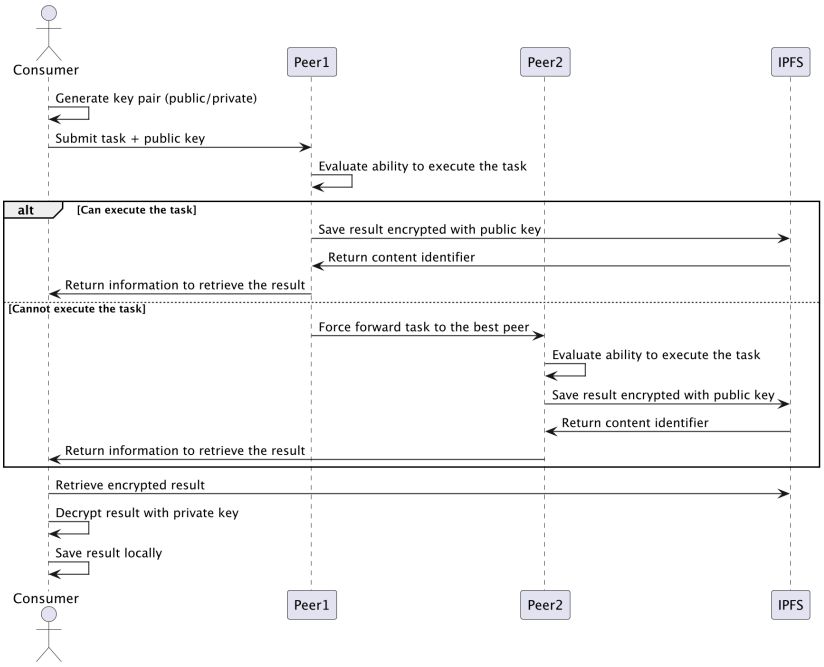

Figure 3.2: Task Submission Flow

Task Submission Flow

The task submission flow is another important aspect of the system, as it allows consumers to submit tasks to the network for execution. This process involves several steps, including task creation, submission, evaluation, execution, storage, and result retrieval. What is important to underline here is that in the analysis phase the focus was on building a structured and secure flow, but the system can be easily extended to support more complex tasks.

Moreover, the workflow itself can be improved, for example, by introducing a task validation and not only the evaluation phase to ensure that the task is valid and safe to execute for the hardware.

As the figure 3.2 shows, since the beginning, the focus has been on building a structured and secure flow. In this case, a generated key pair for each task is used to add an extra layer of security to the task result storage ensuring confidentiality.

Two more components are needed to complete the flow: the **Evaluator** 4.13 and the **Executor**. The Evaluator is responsible for deciding if a node is suitable to execute a task, based on its capabilities and available resource metrics. Another particular step to note is the redirection of the task

that will be analyzed and discussed in the Testing and Evaluation Chapter 5.

The Executor, instead, is responsible for executing the task on the node. While for this prototype, the provided executor is basic, in the future, more advanced executors can be implemented to support different types of tasks and computations. Imagine a scenario where the system supports scientific computing, AI, or privacy-preserving analytics, each requiring specialized executors to handle specific workloads or a community of devs that can contribute to the system creating general purpose executors.

Another aspect that can be deepened is the creation of a specific DSL (Domain-Specific Language) to allow users to define custom tasks, computations or scheduling policies.

Selection algorithm

The selection algorithm is another main part of the system, as it determines which node is suitable to execute a task. It is part of the business logic of the Evaluator. In this case, like for the Executor, a basic selection algorithm has been implemented, but more advanced algorithms can be developed to support different use cases or scenarios. Specifically, this evaluator checks who are the node with the lowest load and the highest capabilities to execute the task to ensure that the task is executed efficiently.

A potential issue that may arise in the system when a provider node, responsible for evaluating a task (i.e., determining whether it can execute it), possesses outdated knowledge about other nodes in the network. If this inconsistency is not properly managed, it could lead to inefficient task execution.

Several solutions can address this problem, ranging from reducing the update interval (high latency problem) to introducing a dedicated node, referred to as the **Maintainer**, responsible for ensuring network-wide consistency of node states.

Adopting this more structured and elegant approach enhances system efficiency, particularly during peak loads or heavy task execution scenarios. Furthermore, this solution improves scalability by allowing a higher number of concurrent tasks to be processed effectively.

Further discussion on this topic, along with a possible reward mechanism to transition towards a broader ecosystem, can be found in Section 5.3. Anyway, in the next chapter, these concepts will be revisited and explained

technically.

3.4 Analyzer Prototype

The second prototype developed in parallel with the main one is the Analyzer. This prototype has been developed to perform measurements on the Tor network and analyze the performance of the system. Since Tor is the network exploited to guarantee the anonymity of the system, it is important to understand how it behaves in different scenarios 4.7.2.

The Analyzer prototype is a simple tool that allows users to measure the latency between nodes communicating through the Tor network. This choice has been made to understand how this network behaves in different scenarios and to measure performances of the system as can be seen in the Testing and Evaluation Chapter 5.

In this section only the main functionalities are explained, but no deep analysis of the implementation is done. I was interested specifically in the Round Trip Time (RTT) between geographically distant nodes, as this metric is fundamental for understanding the network's performance and latency. This is not a primary problem in the system but is something affecting the performance in its overall and needs to be analyzed. So, to estimate the latency between nodes, the Analyzer prototype has been developed. In the Figure 3.3 the main actions done by the Analyzer are shown.

The average Round Trip Time (RTT) is determined by transmitting a series of packets between two nodes, with different Exit and Entry nodes specified in the *torrc* (Tor configuration) file. This approach has proven valuable in assessing task submissions within the system and analyzing how Tor network latency varies across different time periods and regions. A detailed analysis of how the average RTT is calculated and worst-case scenarios can be found in the Testing and Evaluation Chapter 5.

It is important to note that the actual computation time of the task is not considered in this analysis.

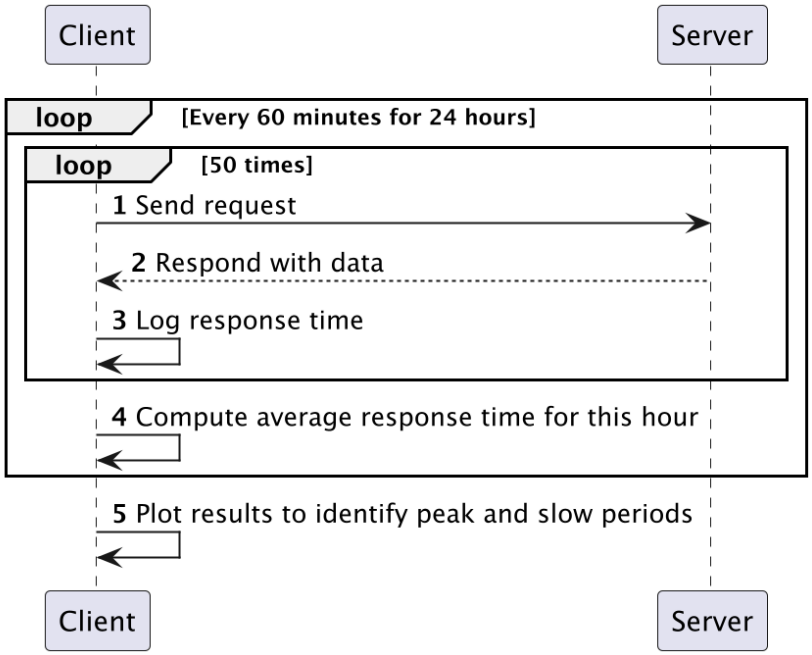

Figure 3.3: Analyzer Prototype

Chapter 4

Design and Implementation

As mentioned in the previous chapter, the main goal of this thesis is to design and implement a system that enables the anonymous computation paradigm. In this section, it is described the design and implementation of the prototype system, focusing on the main parts, the used technologies and the design choices made. Moreover, at the end of this chapter, a description for a second prototype developed to measure the performance of the system is provided.

Before starting with the technical details, it is important to establish a glossary of terms that will be used in the following section. To avoid misunderstandings, the ubiquitous language [4] is reported in Table 4.1.

4.1 Scenarios

The last section before starting with the very technical details is dedicated to the scenarios in which the system can be used to create an idea of the possible applications of the system.

4.1.1 Secure Data Processing

An example can be investigative journalists that often analyze leaked documents containing sensitive information. Using traditional cloud services or personal infrastructure risks exposing sources and data to surveillance, censorship, or retaliation. In this case, a user can be a journalist, a whistle-blower, a researcher or a citizen that wants to analyze some data without revealing their identity.

Terms	Meaning	Synonymous
Node	A participant in the network	Peer
Bootstrap Node	A node that helps other nodes to join	Seed
Provider Node	A node that performs computations	Worker
Consumer Node	A node that submits tasks	Client, Submitter
Domain Event	A significant occurrence in the system	Event
Task	A computation to be performed	Job
Task Result	The output of a computation	Output
Task Type	A category of computations or jobs	Job Type
Submission	The act of sending a task to the network	Request
Force Submission	A task that must be completed	Mandatory Task
Result	The output of a computation	Output
Executor	A component that performs specific task type	—
Evaluator	A task evaluator based on some criteria	—
Metric	Resource usage or performance indicator	Resource Metric
Discovery	The process of finding nodes in the network	Lookup
Encryptor	A component that encrypts or decrypts data	—

Table 4.1: Glossary of Terms for the System

Splitting the entire job in different tasks and distributing them to different nodes in the network, the user can be sure that no single node has access to the entire dataset and that the results will be returned securely. The involved actor retrieves the results without revealing their identity, location or being censored.

4.1.2 Censored Regions

Activists, researchers, and citizens in restrictive regions face heavy internet censorship and surveillance, preventing them from running computational tasks related to encrypted messaging, digital forensics, or decentralized finance. In this way, a real community can be created, where users can share their computational power and use the network to run their tasks without any censorship.

4.1.3 General Scenarios

Anonymous computing enables a lot of different scenarios, and it can be used in various contexts.

1. **Investigative Journalism**: Journalists can analyze sensitive data without revealing their identity.
2. **Secure Collaboration**: Whistleblowers and researchers can collaborate safely in hostile environments.
3. **Censorship Resistance**: Activists can process and analyze encrypted communications freely.
4. **Secure Data Sharing**: Users can share sensitive data without revealing their identity.
5. **Free Speech**: Users can analyze and process data without censorship or create new applications that cannot be censored.
6. **Decentralized Financial Computation**: Users can run specific tasks to deploy some smart contracts or to analyze some financial data without revealing their identity.

This system can be used in various scenarios, and it can be easily extended to support new use cases.

4.2 Business Requirements

Now that with the scenarios the ideas in which the system can be used have been clarified, it is time to define the business requirements that the system must satisfy.

Since the idea behind the system is new, the design should consider ease of extensibility and scalability, allowing changes and improvements to be made without affecting the core functionalities. In this specific case, writing some good business requirements that include all software engineering best practices keeping in mind the security, anonymity and decentralization nature of the system has been challenging.

The problem encountered during the analysis phase was to find a balance between an open and easy-to-extend system and a concrete proof of concept

that could be implemented in a reasonable amount of time. This has been done following the best practices of software engineering and a Dev/Ops approach to the development as described in the following sections.

The section has been divided into provider and consumer parts, and the *use cases* formalism has been used to describe the requirements [5].

4.2.1 Provider

A provider node is a node that performs computations and returns the results to the consumer node who has submitted the task, but before it has to join the network and be able to receive tasks.

1. **Joining the network**:
   (a) **Actor:** Provider Node
   (b) **Precondition:** The provider node has been started and is ready to join the network knowing the address of a bootstrap node.
   (c) **Post condition:** The provider node is part of the network and can receive tasks.
   (d) **Flow:** The provider node sends an event message to the bootstrap node to join the network.
   (e) **Main Success Scenario:** The provider node receives a response from the bootstrap node with some random selected peers and is part of the network.
   (f) **Extensions:** The provider node cannot join the network because the bootstrap node is not reachable. It tries again after a certain amount of time or try to join through another bootstrap node.
2. **Discover Nodes**:
   (a) **Actor:** Provider Node
   (b) **Precondition:** The provider node is part of the network.
   (c) **Post condition:** The provider node knows some peers in the network.
   (d) **Flow:** The provider node sends a *Discover Event* to the network to discover some peers with a specific interval (according to Distributed Hash Table (DHT) principles).

(e) **Main Success Scenario:** The provider node receives a response from some peers in the network.

(f) **Extensions:** The provider node cannot discover any peers in the network.

3. **Metric Sharing**:

(a) **Actor:** Provider Node

(b) **Precondition:** The provider node is part of the network.

(c) **Post condition:** The provider node has shared its metrics with the network.

(d) **Flow:** The provider node sends a *Metric Available Event* Message to the network to share its metrics with the network.

(e) **Main Success Scenario:** The provider node receives a response with an acknowledgment and the other (if known) peer metrics.

(f) **Extensions:** The provider node cannot share its metrics with the network. It retries after a certain amount of time.

4. **Task Receiving**:

(a) **Actor:** Provider Node

(b) **Precondition:** The provider node is part of the network.

(c) **Post condition:** The provider node has received a task from the network.

(d) **Flow:** The provider node receives a *Task Event* from a Consumer Node.

(e) **Main Success Scenario:** The provider node receives the task and starts the evaluation.

(f) **Extensions:** Something is failed during the task receiving. The task is considered failed.

5. **Task Evaluation**:

(a) **Actor:** Provider Node

(b) **Precondition:** The provider node is part of the network and has received a task.

(c) **Post condition:** The provider knows if can execute the task or not.

(d) **Flow:** The provider node evaluates the task with a specific Task Evaluator, based on a defined algorithm.

(e) **Main Success Scenario:** The provider node can execute the task and start the computation.

(f) **Extensions:** The provider node cannot execute that task because some other peers have better metrics. It redirects the task to the other provider node forcing it to execute the task.

6. **Task Execution**:

(a) **Actor:** Provider Node

(b) **Precondition:** The provider node has received a task from the network and has evaluated it.

(c) **Post condition:** The provider node has executed the task, stored the result and returned the content identifier to the consumer node.

(d) **Flow:** The provider node executes the task, stores the result on IPFS and sends the content identifier to the consumer node.

(e) **Main Success Scenario:** The provider node starts the computation.

(f) **Extensions:** The task execution fails, and an error is sent to the consumer node.

7. **Task Result Storage**:

(a) **Actor:** Provider Node

(b) **Precondition:** The provider node has executed the task and has the result.

(c) **Post condition:** The result is encrypted and stored on IPFS.

(d) **Flow:** The result is available, and the provider node stores it on IPFS only after encrypting it with a consumer-provided public key.

(e) **Main Success Scenario:** The result is stored securely on IPFS, and the content identifier is sent to the consumer node.

(f) **Extensions:** The result cannot be stored on IPFS, and the task is considered failed.

4.2.2 Consumer

A consumer node is a node that can submit tasks to the network and retrieve the results. Before submitting a task, the consumer node has to know some peers in the network, potentially even a single one.

1. **Submit a Task**:
   (a) **Actor:** Consumer Node
   (b) **Precondition:** The consumer node has a task to submit, a generated key pair to encrypt/decrypt the result and a provider node address.
   (c) **Post condition:** The consumer node has submitted the task to the network.
   (d) **Flow:** The consumer node sends a *Task Submission Event* to the network containing the task details and the public key that has to be used to encrypt the result.
   (e) **Main Success Scenario:** The consumer node receives a response with an acknowledgment and the content identifier of the task.
   (f) **Extensions:** The consumer node cannot submit the task to the network. It tries again after a certain amount of time.
2. **Consult Task State**:
   (a) **Actor:** Consumer Node
   (b) **Precondition:** The consumer node has submitted a task to the network.
   (c) **Post condition:** The consumer node knows the state of the task.
   (d) **Flow:** The consumer consults the state of the task in the network.
   (e) **Main Success Scenario:** The consumer node receives a response with the state of the task.
3. **Consult Task Result**:

(a) **Actor:** Consumer Node

(b) **Precondition:** The consumer node has submitted a task, the task has been completed, and the content identifier for that task has been received.

(c) **Post condition:** The consumer node has the decrypted result of the task.

(d) **Flow:** The consumer node retrieves the task result from IPFS with the content identifier received by a provider node.

(e) **Main Success Scenario:** The consumer node receives the result of the task from IPFS and decrypts it with the private key.

(f) **Extensions:** The consumer node cannot retrieve the result from IPFS. It retries after a certain amount of time.

4. **Task Result Storage Locally**:

(a) **Actor:** Consumer Node

(b) **Precondition:** The consumer node has retrieved and decrypted the result of the task.

(c) **Post condition:** The consumer node has stored the result locally.

(d) **Flow:** The consumer node stores the result locally in a secure way.

(e) **Main Success Scenario:** The consumer node has stored the result locally.

4.3 Functional Requirements

The functional requirements are the main part of the system that has to be implemented to satisfy the business requirements. For the same reason as above, they are split into provider and consumer sections.

The functional requirements are written exploiting the user story format, which is a way to describe the requirements in a more human-readable way.

4.3.1 Provider

1. **Discover**:

(a) As a **Provider**
I want to be able to join the network.
So that I can receive and send domain events.

(b) As a **Provider**
I want to be able to discover some peers in the network.
So that I can know some peers in the network.

2. **Task**:

(a) As a **Provider**
I want to be able to receive a task from the network.
So that I can execute it and return the result to the consumer node.

(b) As a **Provider**
I want to be able to execute some type of tasks.
So that I can execute them and support most types of tasks.

(c) As a **Provider**
I want to be able to evaluate a task.
So that I can decide if I can execute it or not.

(d) As a **Provider**
I want to be able to redirect a task to another provider node.
So that I can forward the task to another provider node that can execute it.

3. **Result**:

(a) As a **Provider**
I want to be able to store the result of a task on IPFS.
So that I can return the content identifier to the consumer node.

(b) As a **Provider**
I want to be able to share my metrics with the network.
So that I can choose and be chosen to execute a task by a busy provider.

4.3.2 Consumer

1. **Task**:

   (a) As a **Consumer**
   I want to be able to submit a task to the network.
   So that providers can elaborate the result, and I can receive it from the network.

   (b) As a **Consumer**
   I want to be able to consult the state of a task.
   So that I can know if the task has been completed or not.

2. **Result**:

   (a) As a **Consumer**
   I want to be able to consult the result of a task.
   So that I can retrieve the result from IPFS and decrypt it.

   (b) As a **Consumer**
   I want to be able to store the result of a task locally.
   So that I can analyze it later.

4.4 Quality Attributes

The quality attributes are the non-functional requirements that the system must satisfy to be considered a good system. In this case, the most important quality attributes are security, anonymity, and decentralization.

Moreover, fault tolerance and scalability are also important to make the system reliable and usable in real scenarios.

In the first part of this section, the quality attributes are divided in two sections, one for the runtime and one for the development time.

4.4.1 Runtime Quality Attributes

1. **Reliability**: The system should be modular and reliable. The system should work even though some peers are faulty or not deployed.
2. **Scalability**: The system should be scalable and should support many peers.
3. **Anonymity**: The system should be anonymous, preventing data leakage.
4. **Observability**: Due to the decentralized nature and no point of control, each provider or consumer should be observable from only the involved peers.
5. **Lightweight**: The system should be minimal and lightweight to be deployed on commodity hardware (requiring no specific hardware).

4.4.2 Development Time Quality Attributes

1. **Usability**: The system should be usable with a user-friendly and minimal interface.
2. **Maintainability**: The system should be maintainable, with the possibility to fix bugs and add new features without affecting the system's overall functionality.
3. **Deployability**: New versions of the software should be deployed with minimal downtime.
4. **Modifiability**: The system should be modifiable, with the possibility to add new features or modify existing ones without affecting all modules or the overall system functionality.
5. **Isolation**: The components composing the system should be isolated and independent from each other, to avoid cascading failures or bugs.

4.4.3 Quality Scenarios

In this section, quality attributes scenarios are presented following the *Stimulus-Response* formalism. This model is used to describe the system's behavior in different scenarios and to understand how the system should behave in real situations [2].

1. **Scalability**:
   (a) **Stimulus**: An unexpected increase in the number of providers and consumers in the network.
   (b) **Stimulus Source**: Many users join the network simultaneously.
   (c) **Environment**: During peak usage hours or when a new feature is released.
   (d) **Artifact**: The discovery system.
   (e) **Response**: The discovery mechanism should support the increased load without generating too much traffic that can slow down the network.
   (f) **Response Measure**: The system should be able to discover new peers in a reasonable amount of time and without overloading.
2. **Availability**:
   (a) **Stimulus**: A critical failure occurs, and a service becomes unavailable.
   (b) **Stimulus Source**: A hardware failure or a network outage.
   (c) **Environment**: The system is in its normal operational state.
   (d) **Artifact**: The peer that becomes unavailable.
   (e) **Response**: The system automatically detects the failure and retry the communication with the peer for a certain amount of time with increasing intervals.
   (f) **Response Measure**: The system recognizes the failure after the retries and the peer is considered unavailable.
3. **Reliability**:
   (a) **Stimulus**: A software fault occurs in one of the system's services.

   (b) **Stimulus Source**: An unexpected software bug causes a service to fail.

   (c) **Environment**: The system is running in a production environment in its normal operational state.

   (d) **Artifact**: The affected peer(s).

   (e) **Response**: The system automatically detects the failure and restarts or blocks itself, depending on the user choice and trying to maintain overall network stability.

4. **Observability**:

   (a) **Stimulus**: I want to check the discovery state of a provider or a state of a submitted task from a consumer.

   (b) **Stimulus Source**: The monitoring system of each peer or something unexpected in the system.

   (c) **Environment**: The system is running in a production environment in its normal operational state.

   (d) **Artifact**: The peer's monitoring infrastructure.

   (e) **Response**: The system provides real-time dashboards for checking discovery state or task state.

5. **Maintainability**:

   (a) **Stimulus**: A vulnerability is reported.

   (b) **Stimulus Source**: A user submits a vulnerability report or automated tools detect a vulnerability.

   (c) **Environment**: The system is running in a production environment in its normal operational state.

   (d) **Artifact**: The affected component.

   (e) **Response**: New fix, if possible, is applied to the affected component.

   (f) **Response Measure**: The fix is applied and deployed to production following the gravity of the issue.

6. **Deployability**:

(a) **Stimulus**: A new version of the software is ready for deployment.

(b) **Stimulus Source**: The development of the new release has been completed.

(c) **Environment**: The system is in its normal operational state and a new version is ready for deployment.

(d) **Artifact**: The peer that has to be updated.

(e) **Response**: The new version is deployed minimizing downtime and without affecting other peers.

7. **Modularity**:

(a) **Stimulus**: A new feature to a module or an entire module needs to be added to the system.

(b) **Stimulus Source**: A product enhancement, a change in requirements, a bug fix or a new feature.

(c) **Environment**: The system is in its normal operational state, and the new part of the system is ready for deployment.

(d) **Artifact**: The specific module that will be modified and their dependencies.

(e) **Response**: New feature without affecting other modules or the overall system functionality.

In this case, the quality attributes scenarios can be considered too much for a simple prototype, but they are useful to understand the desired system's behavior in real scenarios. They represent a base for the system's testing and validation, and they can be used to expand the system in the future. Some of them, like the scalability and the availability, are fundamental for this type of system, and they have been considered during the development phase. More in the testing chapter can be found about how these scenarios have been tested.

4.5 Architecture

Every node of the system is composed of different components that interact with each other to provide the functionalities described in the previous

sections. To guarantee the system's quality attributes, the system has been designed following the **Clean Architecture** principles [6]. It is a software design methodology that separates the software into different layers, each with its own responsibility, with the main goal of making the software independent of the frameworks and libraries used to develop it.

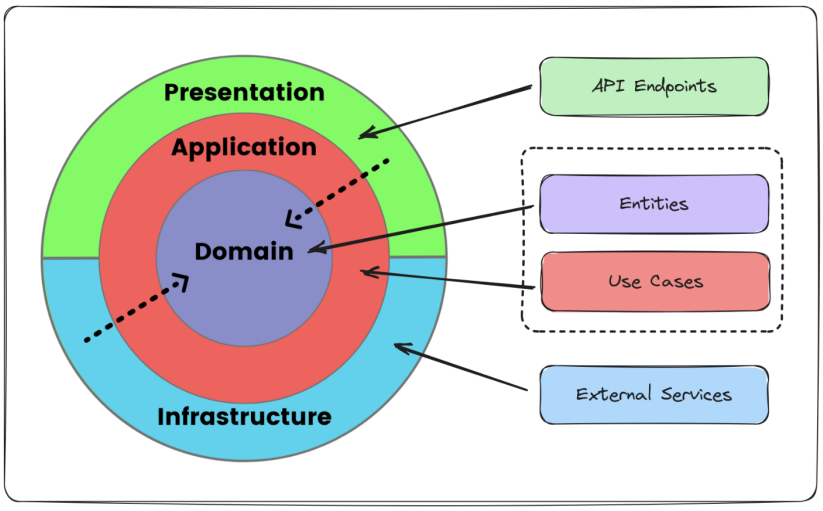

Figure 4.1: Clean Architecture

This architecture favors the separation of concerns and helps to follow the *Dependency Inversion Principle (DIP)* [7] during the implementation phase (Section 4.7.1). Another fundamental aspect is the independence of the layers that helps to maintain a core business logic that can be used in different contexts abstracting away from possible technical issues.

4.5.1 Components

Each node typology has different components that interact with each other to provide the core functionalities [8]. They are different because of the different roles that the nodes have in the network.

Provider Node Components

Each provider node has some subcomponents, one foreach of the main functionalities. As understood, the provider has three main jobs: discovering peers, sharing metrics and executing tasks.

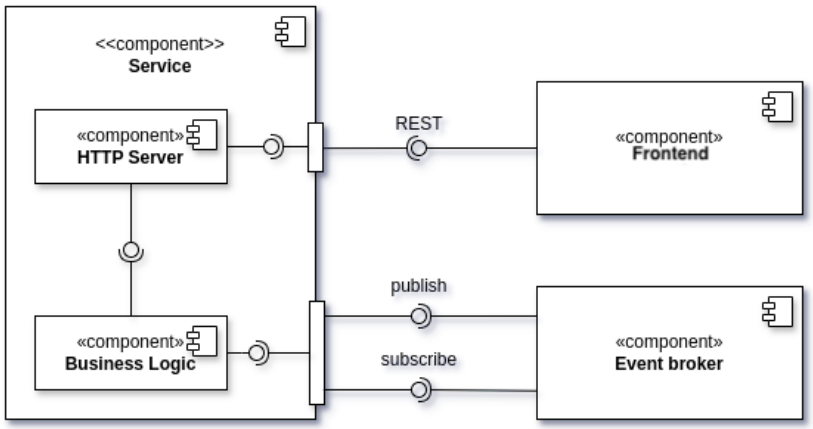

Figure 4.2: Components and connectors

For this reason and for separation of concerns, three different services have been created: the **Discovery Service**, the **Metric Service** and the **Task Service**. Each one inside keeps the specific business logic, and the main component, the **Node Service**, uses them to interact with the network.

The connectors are just a little bit more complex, but what is important to understand here is that they are three: the **Event Broker** to interact with the network, the **Repository Connector** to interact with the storage layer (IPFS in this case), and the **REST API** to let the user monitor the node. The APIs in this case are exposed only internally for security reasons.

Consumer Node Components

The structure of the consumer node is very similar to the provider one considering the connector part but very different in the components. In particular, the components are the same without the **Discovery Service Component** and the **Metric Service Component** that are not needed in this case. Also, these components share only the same names because they have naturally different business logic, simplifying the overall structure of this typology of node.

In this case, the responsibility of the **Node Service** is to interact with the network using the **Task Service** to handle task submission and result retrieval, and the **Event Broker** to facilitate network interactions. This is significantly less work compared to the provider's responsibilities (network discovery, metric sharing, and task execution mechanisms).

Event Broker

Event broker in a decentralized system represents a challenging component. Gossip-based publish-subscribe is the decentralized messaging paradigm that leverages gossip protocols for disseminating events across a network that has been exploited [9]. Unlike traditional broker-based pub-sub systems, where a central broker manages subscriptions and message delivery, gossip-based pub-sub distributes messages in a Peer-to-Peer (P2P) fashion. Nodes periodically exchange information with a subset of their peers, ensuring that events propagate throughout the network in an eventually consistent manner. This approach enhances fault tolerance, scalability, and resilience against failures, making it well-suited for large-scale distributed systems. However, the con in this case is the redundant message transmission that combined with high latency network like Tor can introduces challenges compared to structured pub-sub models.

Figure 4.3: Gossip Pub Sub Mechanism

4.5.2 Other considered alternatives

While the **Event-Driven Model** has been chosen for the system, other alternatives were evaluated, in particular the **Actor Model**.

This model aligns well with the system's requirements. However, its complexity and steep learning curve were considered significant drawbacks looking at the system's future vision of being adopted by a developer community and the need for a prototype to be developed in a reasonable amount of time. Moreover, the Event-Driven Model has been considered more suitable because of its **Loose Coupling** and **Flexibility**, while an Actor Model tends to be more **Tightly Coupled** because of the because of the direct communication between actors (messages), which often requires them to maintain

knowledge of one another's state and behavior. This tight interaction can introduce dependencies that reduce the system's ability to scale independently and adapt to changes in the environment.

From an **ecosystem perspective**, the **Event-Driven Model** was preferred because of its key advantages:

1. **Loose Coupling & Flexibility**: Components remain independent, making modifications and integrations easier.
2. **Stateless and Scalable by Design**: Enables effortless scaling.
3. **Lower Learning Curve**: More accessible to developers already familiar with standard messaging protocols.
4. **Better for Asynchronous Workflows**: Naturally integrates with async/await patterns and reactive programming.

The **Actor Model**, however, remains a powerful alternative that provides provide fault tolerance and resilient state management.

4.5.3 Patterns

Communication

There are multiple types of communication patterns that are used in this system. Specifically, when a user needs to communicate with its node (or peer) backend to exchange essential information required for completing a business operation, the **Remote Procedure Invocation** (request/response) pattern will be the preferred approach.

This type of communication follows a **one-to-one** model. On the other hand, the **Asynchronous Messaging** pattern (publish/subscribe) will be used in scenarios where a peer needs to notify other components about the occurrence of an event, a state change, or an action taken by a user.

In this case, the communication can be either **one-to-one** or **one-to-many**, depending on the business requirements and the business action to perform.

1. In the case of **RPC**, communication will be implemented using the **REST** mechanism over the **HTTP** protocol.

2. In the case of **asynchronous messaging**, the communication will be event-driven.

All events components and event brokers rely on **socket-based** mechanisms. Moreover, the system is designed to operate in either **anonymous** or **non-anonymous** mode, depending on the specific run and user requirements. To ensure flexible and secure routing of network traffic over Tor, in case of anonymous mode, the system uses the **SOCKS5 proxy** to provide a secure communication channel between peers. More on the technical details can be found in the Section 4.7.2.

Deployment

1. **Node as a Container**: Each node is deployed as a container, ensuring **isolation** and enabling **independent scaling**. Container images are **lightweight**, fast to build, and quick to start. For containerization, **Docker** has been chosen as the primary containerization platform. More details in the Containerization Section 4.9.1.

2. **Externalized Configuration**: Instead of hard-coding configuration values for a specific environment, configuration properties are supplied at **runtime**. The **Push model** approach has been adopted, where configuration values are passed to services through **environment variables** or **configuration files**, ensuring flexibility and adaptability across different deployment environments and demos.

Security

1. **Token Authentication**: The system uses **token-based authentication** as an additional security layer, even though the API is accessible only within the **peer network** and is not exposed externally. Each request must include a **Bearer Token** in the `Authorization` header. For now, the token is only passed to the node as environment variable during the deployment phase, but in the future, it can be generated by an authentication service and passed to the node during the bootstrap phase. When a request is received, the system verifies the token validating it, preventing unauthorized access and reinforcing security within the peer network.

2. **IPFS encryption**: Since the system exploits IPFS for storage, and as introduced in the Background Section 3.2 no default encryption layer is provided, a custom encryption mechanism to guarantee data confidentiality has been implemented exploiting the **Asymmetric Encryption**. By generating a unique key pair for each task, the private key is kept secure and only accessible by the submitter, while the public key, sent with the task details, is used for encryption before the data (task result or whatever) is stored on the public repository. Formally, the process works as follows:

   (a) A public-private key pair is generated when preparing to submit a task.

   (b) The public key is sent with the task to encrypt the result before storing it on IPFS.

   (c) The private key, not shared by the submitter, is the only key by which the encrypted result can be decrypted, following the basic concept of the asymmetric encryption.

   (d) Since only the submitter holds the private key, it is the only entity that can decrypt the result and access the data.

   This design ensures that no third party, including other participants in the system or malicious actors can access or modify the data after its storage without the submitters' private key. It is a simple but effective mechanism to add an additional (in this case necessary) layer of security to the storage system. In the Future Extension Section 5.3 an improvement to this mechanism is proposed.

3. **Secrets**: During the CI/CD pipeline, secrets are passed to the GitHub runner exploiting the GitHub Secrets feature. In this case, a fine-grained token (minimum permissions required) is passed to the runner to automate some GitHub Actions job, like for example the release of a new version of the software. This type of feature is useful to avoid hardcoding secrets in the codebase and to keep them secure also in a CI/CD pipeline for a public repository. The same feature has been exploited to automate the Docker image push to the Docker Hub Registry.

4.6 Detailed Design

In this section, the detailed design of the system is presented.

4.6.1 Packages

Each provider and consumer node is composed of different packages, structured basing on the clean architecture principles.

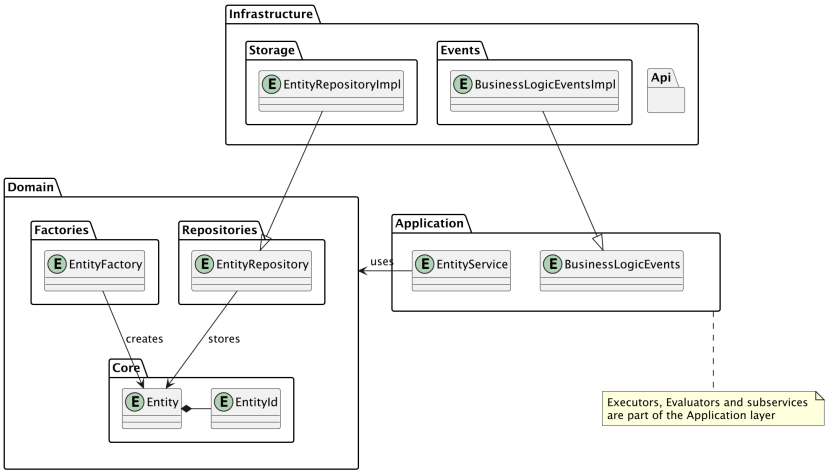

Figure 4.4: Packages Structure

4.6.2 Domain Events

The system follows an event-driven model, and some events like **TaskEvent** are shared between the provider and consumer nodes. In this case, a **Shared Kernel** has been created to store the domain events that are shared between the two nodes. Here a particular design choice that has been made was to create interfaces for basic abstraction, such as **DomainEvent**, to make event management within the domain even more flexible and modular, keeping clear the concept also for new features or devs.

Provider Events:

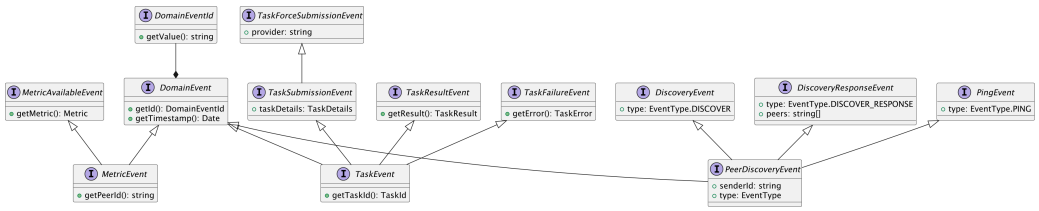

Figure 4.5: Provider Domain Events

Consumer Events:

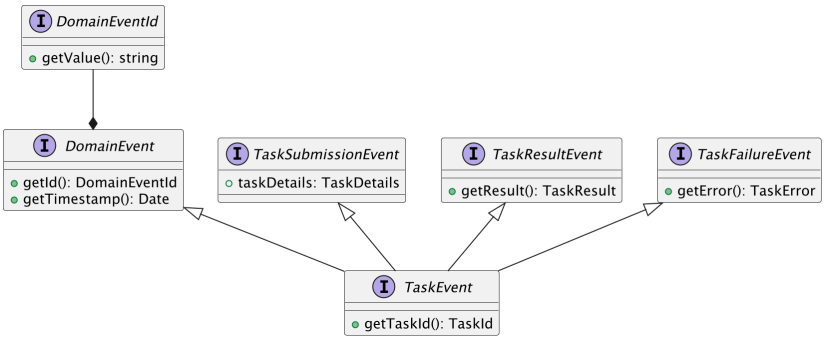

Figure 4.6: Consumer Domain Events

These events are documented using the **Async API** specification. As can be read in the Dev/Ops Section 4.8, the documentation is automatically generated and published on the GitHub Pages of the repository. In this case accepted payloads, examples, and responses are documented to make the system more understandable for possible new developers and to make the system more usable for the user.

The same approach has been used to document the REST API (following **OpenAPI** specification). The documentation is available on the GitHub Pages of the repository that can be found here. Direct link to AsyncAPI and OpenAPI (provider, consumer) documentation.

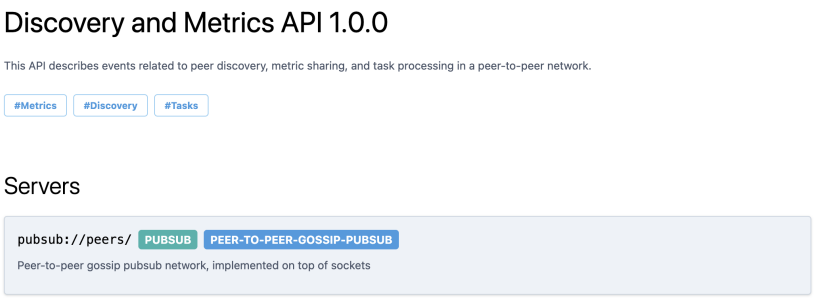

Figure 4.7: Documentation Example

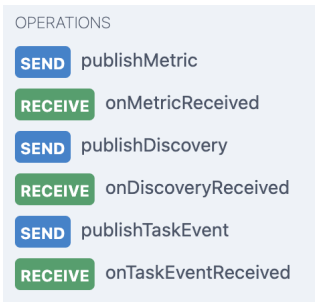

Figure 4.8: Handlers Example

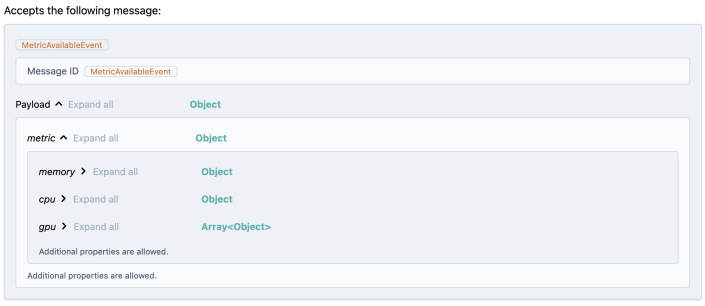

Figure 4.9: Payload Example

4.6.3 UML Diagrams

In the next sections some UML diagrams are presented to show the structure of the provider and consumer node respectively. In particular, every subsystem is described with a class diagram with some notes to explain the main functionalities. Some implementation of the main interfaces does not repeat all the methods for simplicity, but they are all implemented in the code.

4.6.4 Provider Node

Discovery Service

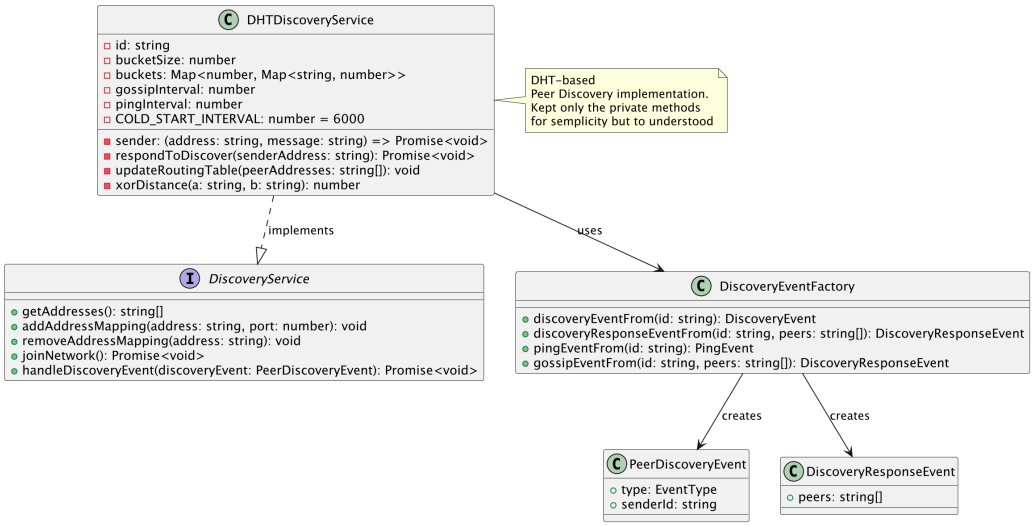

Figure 4.10: Provider Discovery Service

Metric Service

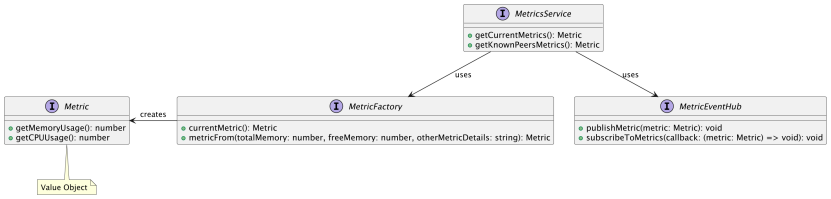

Figure 4.11: Provider Metric Service

Task Service

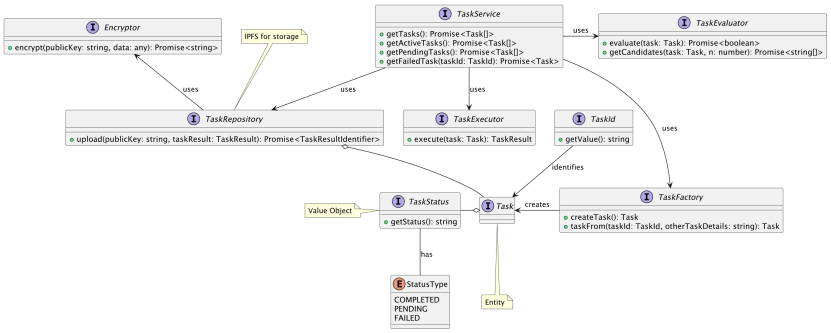

Figure 4.12: Provider Task Service

An important note here is the definition of the **Evaluator** that is used to evaluate the task and to understand if the node can execute it or has to redirect it to another node. As seen in the *Task Service* diagram Figure 4.12, the **Task Evaluator** is a component used by it. It is a simple interface that has to be implemented by the user to define the evaluation logic, supporting extensibility but fixing the business logic.

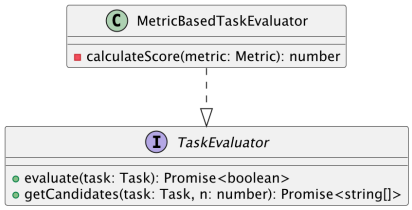

Figure 4.13: Provider Task Evaluator

4.6.5 Consumer Node

Consumer Task Service:

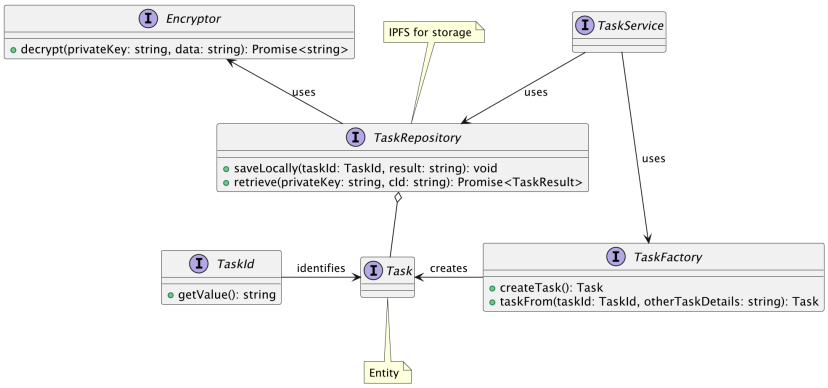

Figure 4.14: Consumer Task Service

As seen in the *Consumer Task Service* diagram Figure 4.14, the **Task Service** is the main component of the consumer node used by the **Node Service** to interact with the network. In this case, the service is more simple than the provider one, and it has complementary functionalities. An important design choice that fixed the business logic is the **Encryptor** that can be used only to decrypt the task result.

4.7 Implementation

4.7.1 Dependency Inversion Principle

From the detailed design section, the system exploits a lot of interfaces fixing the business logic and the separation of concerns also before the real implementation. One of the principles that have been followed is the **Dependency Inversion Principle** that is a key principle of the Clean Architecture.

It is a concept in software design that helps keep different parts of a system loosely coupled. It states that high-level modules (which define business logic) should not directly depend on low-level modules (such as databases or network connections). Instead, both should rely on abstractions.

Improving **flexibility** and **maintainability** without impacting the core logic. Another pro is that this principle makes it easier to test the codebase as

it allows for the use of mock objects to simulate dependencies during testing, a feature that is particularly useful in possible future testing phases.

4.7.2 Technologies

Programming Language

The programming language chosen for the implementation is **TypeScript**, a superset of JavaScript that adds static typing to the language. This choice has been made to ensure a more robust codebase and to leverage the benefits of static typing during the development phase. Moreover, TypeScript is widely used in the industry and has a large community, making it easier to find resources and support, especially for a project that aims to be open-source.

Some cons of TypeScript are the learning curve and the compilation time that can be longer than JavaScript, but the benefits in terms of code quality and maintainability outweigh the cons. In this case I was also familiar with TypeScript, so the choice was natural. This choice is not a constraint for the possible creation of other typologies of nodes because once the interfaces and communication patterns are defined, the implementation can be done in any language.

Express

Express is a minimal and flexible Node.js web application framework that provides a robust set of features for web apps. In this project, Express is used to create the REST API layer that exposes the system monitoring functionalities to the user. The choice of Express was made because of its simplicity, making it ideal but not for any particular constraint.

Tor

Tor (The Onion Router) is a privacy-focused network that enables anonymous communication by routing traffic through a series of encrypted relays. It adds an extra layer of security by encrypting data multiple times, ensuring that the data inside the network remains hidden. In this system, Tor is used to anonymize communications between nodes, ensuring that the identity and location of participants remain undisclosed. Tor offers **Hidden**

Services that allow services to be hosted on the tor network. The choice of Tor was made due to its strong anonymity guarantees.

An **Hidden Service** is a service that is only accessible through the Tor network. It is a server that is configured to receive inbound connections only through the Tor network, ensuring that the server's location remains hidden.

It offers a series of advantages that are fundamental for this system:

1. **Anonymity:** The IP address of an onion service remains hidden, protecting both users and operators.

2. **Encryption:** All traffic between users and the onion service is encrypted at the network level. However, it is important to note that this encryption applies only to the transport layer within the Tor network. In general, once the data reaches the onion service, its security depends on how the service processes and stores it. If the service does not implement additional encryption or security mechanisms, sensitive information could still be exposed or vulnerable to attacks on the server side.

3. **Automatic Address Generation:** Users do not need to buy domain names; onion addresses are cryptographically generated.

4. **Integrity Assurance:** The .onion URL itself ensures users connect to the correct service without risk of tampering.

To access an onion service, users must know its .onion address, which consists of 56 alphanumeric characters followed by `.onion`. Additionally, websites can implement the `Onion-Location` header, allowing automatic redirection to their onion counterpart.

A possible enhancement involves authenticated onion services, which require users to provide an authentication token before gaining access. This strengthens security by restricting access to authorized users only. Tor Browser supports this mechanism, allowing users to enter a private key when prompted. An encountered problem while the containerization process of the system inherent to how Hidden Services are managed by Tor can be found in the Deployment Chapter 4.9.1.

Entry Nodes (or **Guard Nodes**) are the first relay point your traffic passes through when connecting to Tor. They are responsible for establishing the

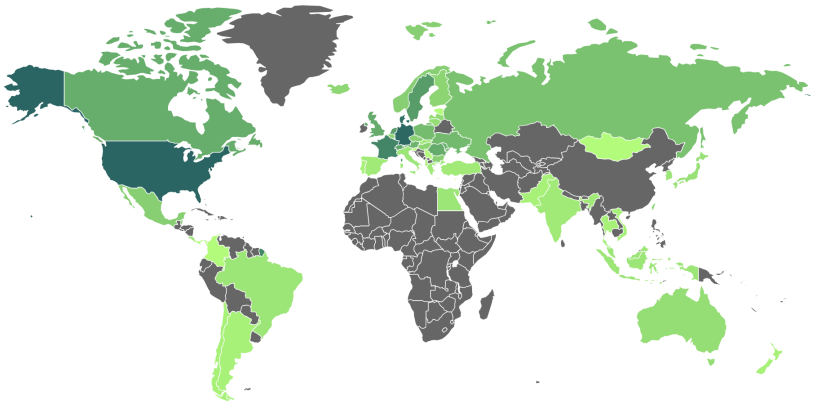

Figure 4.15: Mapped Exit node

connection between your device and the Tor network, providing anonymity by masking your real IP address.

Exit Nodes are the last relay in the Tor network that your traffic passes through before reaching its final destination. They are responsible for sending the encrypted traffic and decrypting the final layer of encryption. However, the exit node can see your traffic's destination but not its source (IP address). An updated list of the Tor Relays and a lot of interesting metrics can be found here. The link to the full Tor technical documentation can be found here. Each process like the circuit creation, the connection to the network, and the hidden service creation is explained in detail in the technical documentation. More on how circuits are exploited for the system can be found in the Testing and Evaluation Chapter 5.

IPFS

IPFS (InterPlanetary File System) is a distributed, peer-to-peer network for storing and sharing data in a decentralized manner. Instead of relying on a central server, IPFS uses content-addressing, where each file is identified by its cryptographic hash, ensuring immutability and resistance to censorship. In this system, it is used to store and retrieve data efficiently avoiding reliance on centralized storage solutions. This choice was made because of its decentralized architecture and data persistence.

The problem encountered here is that, as mentioned, this technology ex-

ploits content-addressing, so once the data are stored on IPFS they became immutable. Moreover, the stored data are not encrypted by default, so it is necessary to encrypt them before storing to ensure privacy and confidentiality. In this system, it is used to persist the info about tasks and tasks results. The design choice to avoid the storage of clear data on IPFS is the generation of a key pair by the consumer node that is used to encrypt the result before the storage. Mechanism deepened in Security Patter Section 4.5.3.

Pinata is a cloud-based service that simplifies storing and managing files on IPFS (InterPlanetary File System). It has been used for simplicity and to avoid the need to run a self-hosted IPFS node, considered time consuming for a prototype. It provides a web interface, API access, and pinning services and some technical documentation that helps the developing process.

Unlike self-hosted IPFS nodes, Pinata is ideal for who don't want to maintain their own infrastructure. In this case, besides the fact that the system is a prototype and a self-hosted IPFS node would be excessive, it was very interesting to understand how it works. This technology is widely used in NFTs, Web3 applications, and decentralized storage solutions, offering scalability, access control, and analytics.

Beyond this, however, the main purpose of the system would require one or more independent nodes to maintain the system decentralized.

SOCKS5 Proxy

SOCKS5 is a networking protocol that routes network packets between a client and a server through a proxy server. In this system, every communication is tunneled through a SOCKS5 proxy to ensure anonymity and privacy when *Anonymous Mode* is enabled. As mentioned in the Background Section 3.2 the choice of SOCKS5 was made because of the Tor network's support for this protocol, making it ideal for routing network traffic through that network. Moreover, this type of proxy supports various protocols, and with this fifth version, also the UDP, that is a nice feature. The problem is that the UDP protocol is not supported by the Tor network, so it is not used in this system.

Briefly, this proxy mechanism is very powerful; it allows the system to transparently route packets through an intermediate relay, preserving anonymity while maintaining efficient communication.

General key advantages of using this proxy include:

1. **Anonymity and Privacy**: When running in anonymous mode, all network traffic is routed through the SOCKS5 proxy.

2. **Protocol Agnosticism**: Unlike HTTP proxies, SOCKS5 supports various protocols.

3. **Flexible Traffic Routing**: The system can dynamically switch between direct connections and proxy-based routing, depending on whether anonymity is required.

4. **Reliability in Restricted Environments**: SOCKS5 helps bypass network restrictions, making it possible to maintain peer communication even in constrained environments.

With its **5h** version, instead, the resolution of DNS queries is possible at the proxy level. In this specific case, the system uses the Tor network, which already provides DNS resolution, so this feature is not exploited. In general, this feature can enhance anonymity and can bypass DNS censorship or avoid DNS leaks.

An interesting feature, according to the Tor Documentation, is that adding the `SafeSocks 1` directive in the `torrc` file prevents DNS leaks by ensuring that all DNS requests are routed through the Tor network. This configuration rejects any connection attempting to resolve DNS outside of Tor, safeguarding the user.

In the implementation, all peer communications for **event-driven messaging** are transparently tunneled through the SOCKS5 proxy when operating in anonymous mode. In this system the proxy is provided directly by the Tor installation. Below, a code snippet shows how the system establishes a connection to a peer through the proxy.

```
1    await SocksClient.createConnection({
2        proxy: {
3          host: socksHost,
4          port: socksPort,
5          type: 5
6        },
7        command: 'connect',
```

```
8           destination: {
9             host: '{address.onion}',
10            port: {port}
11          }
12        })
```

To use the SOCKS5h version, the `type` field in the proxy configuration object must still be set to the value 5 but a Lookup Function must be provided to set the DNS lookup to the proxy server. This depends on the library used to establish the connection, in this case the **socks** modules has been exploited.

Docker

Docker is a containerization platform that allows applications to run in isolated environments, ensuring consistency across different systems. It enables the packaging of software and its dependencies into lightweight containers, making deployment more efficient and reproducible. In this system, Docker is used to encapsulate various components, ensuring that they run in a controlled and consistent manner across different machines. The choice of Docker was made to simplify deployment, enhance portability, and manage dependencies efficiently.

In this particular system, the exploitation of Docker *Volumes* brings the possibility to store the data outside the container, ensuring data persistence also for a consumer that want to submit a tasks always using different different Hidden Service onion address. This is possible because the system is configured to store the hidden service private key in a volume that is shared with the container. For more explanations about job unit submissions return to the Data Flow Analysis Figure 3.2 while for the specific containerization process, where this feature is exploited, continue to the Deployment Chapter 4.9.

4.7.3 Transport Layer

One of the main challenge in this system was to ensure secure and anonymous communication between peers. In the early stage of the implementation, after fixing the business logic, architecture and the main components, the focus was on searching some good library that can help to achieve this

prototype. The attention was in particular to **libp2p.io**, a modular network stack that can be used to build peer-to-peer applications. This library has gained popularity and is now used and affirmed in the peer-to-peer community. This modular and extensible framework adapted across many programming languages (JavaScript,Go, Rust, C++, Nim, Java/Kotlin, Python, Swift, ...) has facilitated its integration into numerous high-profile projects.

For example *Ethereum blockchain, IPFS, Filecoin, Polkadot* and many others are using libp2p to build their network stack.

The ongoing development and adoption of libp2p underscore its critical role in decentralized technologies, making it an ideal choice for this system. The learning curve of this library I think is the only con. While for simple project the library can be easily introduced and used, for complex projects it can be difficult to manipulate and understand all the features to reach the desired result.

Moreover, in this case, the library is not designed to be used on top of the Tor network so it necessitates some additional workaround to make it feasible.

The initial tries aimed to use this library in a "hacky" way (modifying directly the library code, not considered correct from the design point of view) were aborted because of the *multiaddr* [1] resolution. Furthermore for each update or fix of the library it was necessary to manually modify involved files or keep a fork of the library with all security implications. This goes against all good security and software engineering best practices and for this reason this line of development was abandoned.

So, after discarding this possibility, an analysis and a feasibility study was conducted to implement a new module for the library that could extend the functionalities to support communications over the Tor network.

What emerged is that the **Transport Interface** needs to be implemented to make a new compliant module (like js-libp2p-websockets). In this case I would have liked to contribute to the library, but the time was limited and the complexity of the task was high. I should have investigated better where the library uses the UDP protocol to generate similar components that encapsulate this UDP traffic in TCP, possible on Tor. Moreover, given the different nature of the two networks, it would have been necessary to

[1] Multiaddr is a flexible and multi-protocol address format used for identifying peers in a decentralized, peer-to-peer network. It allows a single identifier to represent a peer using multiple transport protocols and addresses.

implement a mechanism to manage the different latencies, timeouts, name resolution mechanism and others in a cascading way. Given the limited time and the fact that this development line could not yet lead to a concrete result, the decision taken was, with regret, to continue discarding also this possibility and using native sockets for communication between peers. Anyway, this has been a very interesting part of the development, and it has been a great opportunity to understand better how the library works and how it can be extended. Surely, more work and study are needed to make this library (or at least some components) work on top of the Tor network, but the potential I think is very high.

Another consideration regarding this possible development is that if I could have used the library I could have focused on an even more complex system, moving towards an ecosystem concept. This however I think is normal in a prototype, where the focus is on the main abstractions and the quality of the architecture, and not on the complexity of the system.

Native Sockets

The communication between peers is managed through native sockets. The system uses the **Node.js** `net` module to establish connections between peers. Also in this case, the transport layer is changed accordingly to the operating mode of the system, in particular the usage of the **SOCKS5 Proxy** to ensure anonymity and privacy or not. To do that, the DIP principle has been followed another time, creating an interface that abstracts the communication layer and implementing two different classes, one foreach mode.

The main problem encountered, in the anonymous mode, is the management of the different latencies or network instability that can be introduced by the Tor network. To solve this problem a retry mechanism has been implemented.

4.8 DevOps

4.8.1 Build Automation

Gradle is used as the build automation tool for the project. It allows for the automation of the build process, including compiling, testing, and packaging the application. The build process is defined in the `build.gradle` file,

which specifies the project's dependencies, tasks, and configurations foreach node. Also the tasks for the code quality check are defined in the build file. In this case, I tried to keep the codebase with only necessary dependencies, so the build process is not something complex.

4.8.2 Version control

DVCS workflow

The project uses Git as the version control system, and my workflow is based on updating the main branch with new features or fixes through dedicated branches. In this way, the main branch always contains the latest stable code. New changes or fixes are introduced through dedicated branches (*feature/name*, *fix/name*, *chore/description*, etc.), ensuring a structured approach to development and a clear history. Releases are directly managed from the main branch when merging a feature (or fix) branch.

Moreover, to ensure consistency and quality, every commit is associated with a meaningful message that describes the changes introduced following the **Conventional Commits** standard.

Semantic Versioning and Release

Both software, the provider node and the consumer one, follow **Semantic Versioning**, with version numbers automatically determined by the CI/CD pipeline. The Semantic Release plugin automates this process. By analyzing conventional commit messages, the plugin determines the next version based on the changes introduced. Additionally, it generates a changelog and creates a new release on GitHub.

This setup can result difficult to understand for a new developer, but it ensures consistency and quality in the release process, making it easier to track changes and manage versions. Potentially, to let the system be more accessible, a more traditional release process could be adopted, but the current setup is more suitable for an open-source project. The only change to do should be the Release process triggered only when a new Pull Request is merged in the main branch and after at least some code reviews.

4.8.3 Quality Assurance

To maintain the quality of the system, some quality assurance practices have been adopted, exploiting these tools:

1. **Prettier**: A code formatter to ensure consistent code style.
2. **ESLint**: A static code analysis tool to identify problematic patterns in the code.

Surely, the system can be improved with more tools and practices, but for a prototype and the fact that I'm the only developer, these are enough to ensure a good code quality and maintainability.

4.8.4 Continuous Integration and Delivery

The CI/CD pipeline is an essential part of the development process, ensuring that the code is continuously integrated, that does not break the build, and that new features are delivered.

1. **Build:** Compiles the project, ensures functionality across Linux, and macOS platforms.
2. **Style Check:** Verifies code formatting and consistency using linters and formatters.
3. **Generate Documentation:** Builds the documentation website and stores it as an artifact for future releases.
4. **Version Calculation:** Determines the next version number for the project based on commit history.
5. **Release Management:** Analyzes commit messages to determine if a new release is required, and if CI/CD file determined conditions are met, tags the repository with the new version number.
6. **Deploy Documentation:** Retrieves the pre-built documentation and publishes it to GitHub Pages.
7. **Docker Image Deployment:** Builds Docker images for all nodes upon release trigger and pushes the new images to Docker Hub with the *latest* tag.

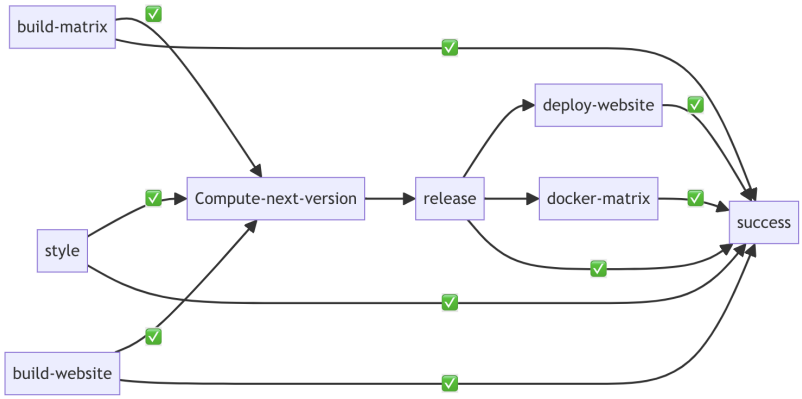

Figure 4.16: CI/CD Pipeline

4.9 Deployment

This section provides a brief overview of the deployment process and the necessary steps to deploy the system but before that, the containerization process is explained.

4.9.1 Containerization

The containerization process is fundamental to ensure a good deployment process. In this case, Docker is used to containerize the system, encapsulating each node in a separate container. Source code of the frontend and the backend are in different project module to keep the system more modular but are packaged together in the same image. This approach has been followed for simplicity and to ensure that every node, including a monitoring dashboard, is encapsulated in a single container. A possible con can be the fact that if some structural or architectural changes are done on thee frontend, the entire docker image needs to be rebuilt.

Dockerfile

The created Dockerfile, as mentioned, builds the application with two main parts: the core of the node and the relative frontend. In this case, a multi-stage build is used to optimize the image size and ensure that only the necessary dependencies are included in the final image.

1. **Base Image:** The build starts from the `node:23-slim` image. This image provides a lightweight Node.js environment.
2. **Step 1: Application Setup**: In this phase, the dependencies are installed, and the application is built.
3. **Step 2: Final Image with Tor Installation**: The final image is created with built files, and Tor is installed and configured.
4. **Step 3: Combining Startup Commands**: The startup scripts are copied, and the startup command is defined.

An important step here is the configuration of Tor, that in this case defines the **Hidden Service** port for each node. Also the proxy port is defined to ensure that the communication between the nodes is tunneled through the relative network.

Multiple improvements can be done in the future, like the creation of a specific user for the application and the definition of a health check for the container.

A problem encountered was that during the first run of the container, the Tor service had not started, and therefore the hidden service address was not defined. This issue was resolved by modifying the startup scripts and the Dockerfile, which now set an environment variable containing the hidden service address.

I said that this problem occurs only the first time, but there is a reason for that. The reason is that the folder in which the hidden service address is stored is a volume, so the data is persisted even if the container is stopped. This design choice ensures that the hidden service address remains consistent across container restarts. However, if the corresponding volume is removed, the hidden service address will change each time the container is started.

Default Port Mapping

Since with this configuration, all the nodes run in the same host machine, a default port mapping was needed. The mapping is done in the Docker Compose file, and it is defined as follows.

Providers (ANONYMOUS_MODE)

1. All providers use an internal `PORT`: **3000**.

2. Their `API_PORT` values start at **4001** and increase sequentially, with mappings formatted as `<API_PORT>:4000`. This ports refers to the port used by the frontend to communicate with the backend. It is important to specify this port, because when thee frontend is served, the requests starrt from outside the container obviously.

3. Frontend ports are assigned sequentially beginning at **8081** (mapped as `<frontend_port>:8080`). This ports refers to the port used by the user to access the monitoring dashboard.

4. An environment variable `BOOTSTRAP_NODE` is set to "true" for the first n providers (where n equals `bootstrap_count`) and to "false" for the remainder. In this case, address of bootstrap node, through thee starting scripts, are shared with the other nodes.

Container names follow the pattern `tor-provider-`i.

Consumers (ANONYMOUS_MODE)

1. All consumers use an internal `PORT` of **3000**.

2. Their `API_PORT` values start at **4901** and increase sequentially, with mappings as `<API_PORT>:4000`.

3. Frontend ports are assigned sequentially beginning at **8181** (mapped as `<frontend_port>:8080`).

Container names follow the pattern `tor-consumer-`i.

Providers (LOCAL_MODE)

1. Each provider receives a unique internal `PORT` starting at **3000** (e.g., provider-1 gets 3000, provider-2 gets 3001, ...).

2. Their `API_PORT` values begin at **4000** (e.g., provider-1: 4000, provider-2: 4001, ...) and are mapped directly (e.g., `"4000:4000"`).

3. Frontend ports are assigned sequentially starting at **8080** (e.g., provider-1: `"8080:8080"`, provider-2: `"8081:8080"`, etc.).

Container names follow the pattern `local-provider-`i, and note that in this case the bootstrap node variables are prefixed with `LOCAL_`.

Consumers (LOCAL_MODE)

1. Each consumer receives a unique internal `PORT` starting at **3900** (e.g., consumer-1: 3900, consumer-2: 3901, ...).

2. Their `API_PORT` values begin at **4900** (e.g., consumer-1: 4900, consumer-2: 4901, ...) and are mapped directly (e.g., `"4900:4900"`).

3. Frontend ports are assigned sequentially starting at **8085** (e.g., consumer-1: `"8085:8080"`, consumer-2: `"8086:8080"`, etc.).

Container names follow the pattern `local-consumer-`*i*.

It is a little bit complex to understand initially, but the main idea is to have a default port mapping that can be easily changed. Another important aspect is the exploitation of the Docker internal DNS that permits to have a communication between the containers using the container name as the address.

N.B. The scripts deepened in the next section follows this logic.

4.9.2 Docker Compose

The most convenient deployment mode of the system is to use Docker containers, in particular exploiting Docker Compose to manage the orchestration of the different services. In this way, another isolation layer is added, and the system can be deployed on any machine that supports Docker. This method brings a lot of advantages, in the development phase, the system can be tested on different machines without any problem, and in the production phase, the deployment is simplified and the system can be easily scaled.

Prerequisite

Docker is the only prerequisite to deploy the system. Keeping the system lightweight and easy to deploy was a key design goal, and Docker was chosen for its simplicity and portability. Moreover, a Docker feature that adapts well to this system is that containers are one-shot, in the sense that they are temporary and can be easily replaced. In this system, this is very useful thinking about Consumer Nodes. A consumer node can be deployed, get a task result, store it locally and then be destroyed, without any problem and only deleting the linked volume. In this way, one shot containers can be used

to improve the system's security. A reference to the deep explanation of this mechanism can be found in the Containerization Section 4.9.1.

4.9.3 Sample Scripts

To streamline the deployment process, several sample bash scripts are provided. Given the decentralized nature of this system, deploying multiple nodes on the same machine is not a logical approach. Considering that this system is a prototype, the scripts are primarily intended for the development phase, where the system can be tested on a single machine. During the production phase, no scripts are required, as the system can be deployed across different machines by simply running the appropriate containers.

While various scripts are available for review and use, the key script for running a demo on your local computer is the *anonymous-shard.sh* script. This script facilitates the creation and execution of a series of containers representing the system's nodes. Once the containers are running, interactions can be made from the Docker host through the corresponding frontend to observe and evaluate the system's behavior.

Usages

To use the script, follow these steps:

1. **Clone**: Clone the repository or download the scripts folder from the repository. Source code is not needed due to Docker images already available on Docker Hub and the externalized configuration that permits to run the system without any code modification. Reference to Containerization Pattern Section 4.5.3 for details.

```
1 git clone https://github.com/paga16-hash/
    anonymous-shard.git
```

2. **Generate**: From the folder, after giving execute permissions to the script, run the following command to generate a test docker compose file with specified number of nodes. With these scripts, the system can be deployed specifying the *ANON_MODE* parameter to *false* if you want to run the system in non-anonymous mode. What is needed

to run the system and in particular to pin files on the IPFS filesystem is a PINATA_API_KEY and a PINATA_API_SECRET that can be obtained from the **Pinata** website for free.

```
chmod +x ./anonymous-shard.sh
./anonymous-shard.sh generate <ANONYMOUS_MODE>
<PINATA_API_KEY> <PINATA_API_SECRET>
<DEV_API_KEY> <NUMBER_OF_BOOTSTRAP_NODES>
<NUMBER_OF_PROVIDERS> <NUMBER_OF_CONSUMERS>
```

3. **Start**: Start the system with the following command. After this command, you can see docker containers running and you can interact with the docker demon or access the frontend to interact with the system.

```
./anonymous-shard.sh start
```

4. **Stop**: Stop the system with the following command.

```
./anonymous-shard.sh stop
```

To interact with the system, you can access the frontend by opening a browser and navigating to `http://localhost:<FRONTEND_PORT>`. Since the script follows the defined port mapping, every node can be accessed using the `localhost` address and the relative port because of the Docker port forwarding. While the system is running, you can submit tasks, monitor the network, and view the results through the frontend.

Other Scripts Since other scripts have been created to automate or simplify the development process, I found useful to provide a brief explanation of them without going into details.

1. **make-requests.sh**: To automate the process of submitting tasks to the system, used to simulate the user interaction with the system and different loads as analyzed in the Testing and Evaluation Chapter 5.

2. **evaluate.sh**: To measure average RTT time between blocks of requests.

These scripts can be found in the `scripts/other` folder of the repository and can be used to automate some processes and to test the system in different scenarios.

4.10 Running System

This section provides only some screenshots of the system running, to give an idea of how the system works.

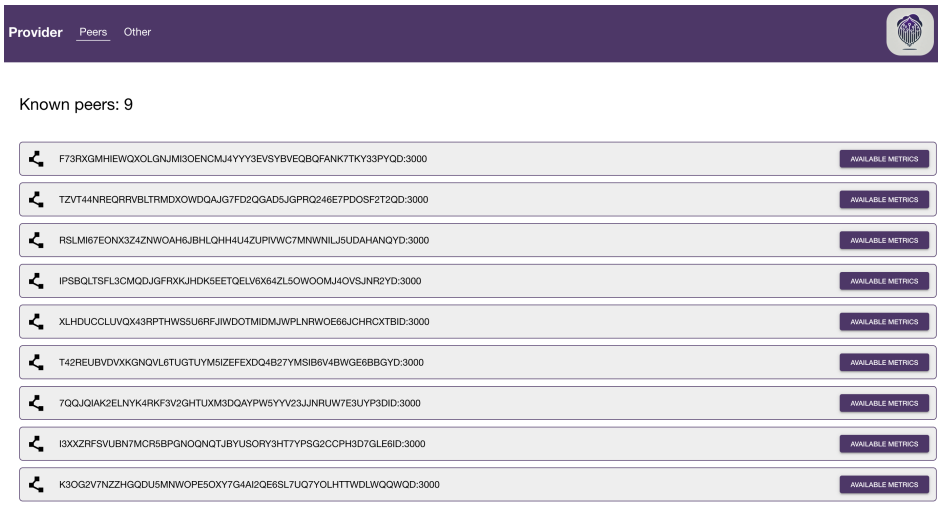

Figure 4.17: Provider Node Dashboard

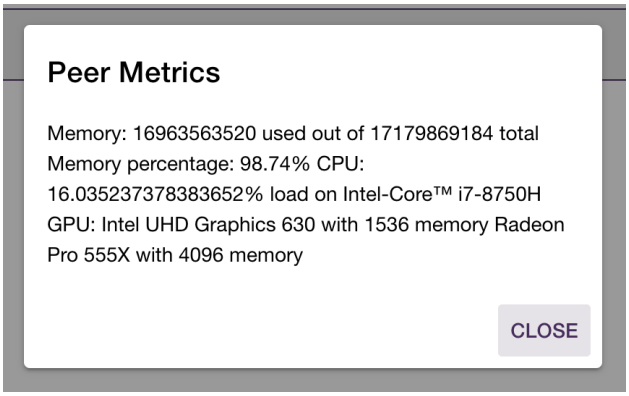

Figure 4.18: Provider Node Metric Consultation

Figure 4.19: Metric in the Backend

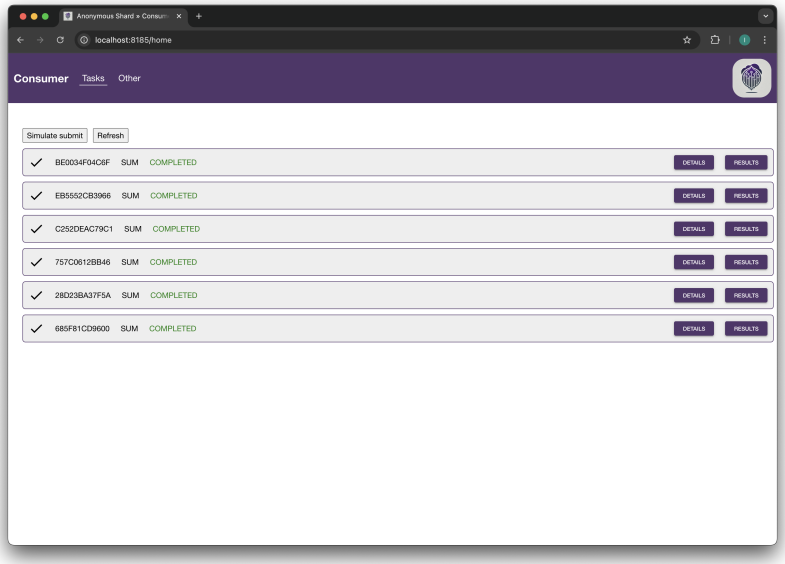

Figure 4.20: Consumer Node Dashboard

Figure 4.21: Pending Task in the Consumer Node

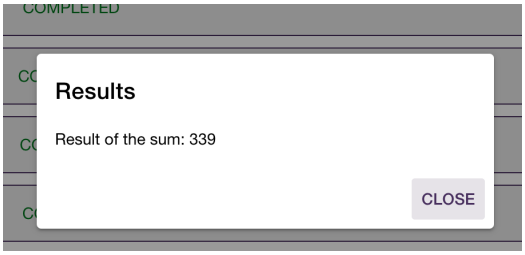

Figure 4.22: Sample Task Result in the Consumer Node

Chapter 5

Testing and evalutation

In this chapter, the tests conducted on the system and the metrics used to evaluate its performance are analyzed. The system has been tested under different configurations to assess various scenarios.

For the first prototype, the tests are mainly focused on evaluating the system's functionalities. In contrast, the second prototype is designed specifically to assess the performance of the Tor network in terms of latency. More important, the second prototype was developed to make general measurements on the Tor network, in this case the following presented measurements are used to understand how the first prototype can behave.

Specifically, the tests aim to determine the effectiveness of the system's scalability, identify potential improvements, or evaluate additional components to handle a large number of nodes. More in general, to assess the feasibility or understand the areas that need further development.

Multiple scalability scenarios have been tested, ranging from a small number of nodes to a large-scale network to observe system behavior in different conditions.

Other tests have been conducted to measure the Round Trip Time (RTT) between nodes in the network. This factor can have a significant impact on message delivery and so on task execution time, specifically within the Tor network. Knowing approximately this value is crucial for evaluating system performance since every or action possibly involves multiple nodes.

Furthermore, all tests are conducted in **anonymous mode**, as performing tests in a non-anonymous mode would not provide meaningful insights given the nature of the system.

More efforts can be made to enhance and refine this evaluation phase as

discussed in the Limitations Section 5.2.5 and in the Future Test Extensions Section 5.3.

5.1 Scalability Test

This test aims to evaluate how the system behaves as the number of nodes increases. The primary focus is to determine the scalability of the network and assess whether an efficient discovery algorithm, such as Kademlia, enables effective scaling. The discovery (explained in Section 3.3.2) and task processing mechanism has been verified for networks of up to 100 producer nodes and 30 consumer nodes on the same machine. The number of submitted task requests was set to 200, and the system was able to handle them without any issues.

A key factor influencing system performance is the tuning of specific parameters. Specifically, since the discovery mechanism uses a hash table, the number of nodes neighbors each node can store in its routing table is one of them. This parameter is crucial, as it directly impacts the number of hops a message must take to reach its destination. Additionally, it affects the time required for node discovery and the overall network stability.

To analyze these effects, the routing table bucket size has been varied from 10 to 70 nodes, observing the system's behavior under different configurations. A frontend view from the point of the *Producers* can be seen in Figure 5.1 and Figure 5.2 where different bucket sizes have been tested.

Another critical aspect that emerged from this test is the importance of the **Metric-Sharing interval**, which determines how frequently nodes exchange resource metrics. This parameter represents a trade-off:

1. If the interval is too long, metric propagation slows down, potentially delaying decision-making and task execution.

2. If the interval is too short, the network may become flooded with messages, leading to unnecessary overhead.

A high Metric-Sharing interval can degrade system performance by increasing task execution times and reducing overall efficiency. As discussed in previous chapters, the component responsible for evaluating task execution feasibility relies on up-to-date resource metrics to make informed decisions.

Since the current *Evaluator* logic as analyzed in the Design and Implementation Chapter 4.6.4 is straightforward and selects the best node for execution based solely on available metrics, outdated information may lead to suboptimal decisions.

The worst-case scenario occurs when the selected provider lacks updated knowledge. If this provider appears to have the best metrics for executing the task but the system is unaware of it, the task is redirected to a suboptimal provider. This introduces a delay due to the need for the task redirection, and since the new provider is not the most suitable, a double additional execution delays occur.

Ultimately, network consistency is crucial: nodes must maintain an updated view of the system's state to ensure efficient resource sharing. If metric propagation is too slow, task execution performance will suffer. To analyze the time needed for specific task execution and the time needed to send a message to another node, a series of tests analyzed in the Average RTT Test Section 5.2 were conducted.

Briefly, important aspects that only regard the network are:

1. The number of nodes in the network.
2. The physical distance between nodes.
3. The number of hops a message must traverse to reach its destination.

Other problems can derive from the execution of the task itself, but in this case where the system is tested in a controlled environment and the tasks were simple, the focus was on the network itself.

5.1.1 Encountered problems

No significant problems were encountered during the scalability tests, as the system was able to handle a large number of nodes efficiently, but some considerations regarding the latency introduced by the Tor network need to be made. The network latency is not a problem of the system itself, and as mentioned, this network, is designed to provide anonymity by routing messages through multiple relays (adding encryption layers), so this behavior is expected. However, it poses a challenge for our system, as it increases the time required for message delivery.

One key observation is that, due to the way the Tor network operates, a new circuit is needed when the previous one is closed due to expiration, inactivity, or other problems. The creation and switching of circuits contribute to the overall latency. This is something to consider when evaluating system performance, as it can introduce significant delays in task execution. In particular, it could justify that cases in which, apparently for no reason, the task execution time is higher than expected.

In this case and in general, if some messages fail to be delivered, or something in the network happens, the retry mechanism is triggered and the communication is re-established. This mechanism is essential to ensure the system's robustness and reliability, as it allows the system to recover from errors and continue functioning correctly. The interval of the retry mechanism is a parameter that can be tuned to balance performance and reliability, or it can be multiplied by the number of retries to avoid network flooding.

Another secondary issue encountered is related to an implementation detail. As the network grows in size, nodes exchange larger packets about metrics or neighbors, as discussed in the Discovery Mechanism Section 3.3.2. The problem, which has been promptly resolved, was related to the reconstruction of the packet at the destination node when a packet was fragmented. This issue was solved by prefixing the packet length at the beginning of the packet, allowing the receiver to determine when the packet was fully received and ready for processing allowing for its reconstruction or not.

The problem has been encountered only when scalability tests were performed because before the local network (so the neighbor packet size) was too small to encounter this problem.

Apart from these two issues, the tests were satisfactory because the system was able to handle a large number of nodes and tasks efficiently. Also, in cases where nodes are manually turned off, the system was able to continue functioning correctly, demonstrating its robustness. In this case, what could clearly be improved and analyzed is the behavior of the network when these nodes are really distributed worldwide. This is a problem that will be addressed in the future and discussed in the Future Test Extensions Section 5.3. The average time for message delivery will be analyzed in the Detailed Analysis Section 5.2.3.

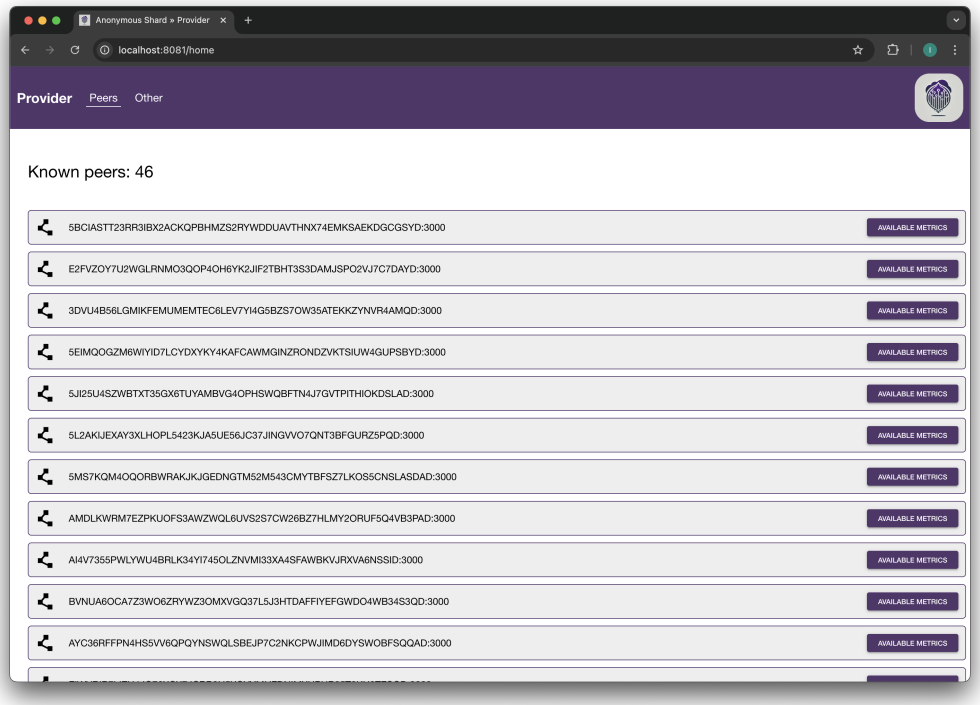

Figure 5.1: Bucket Size 50 reaching saturation

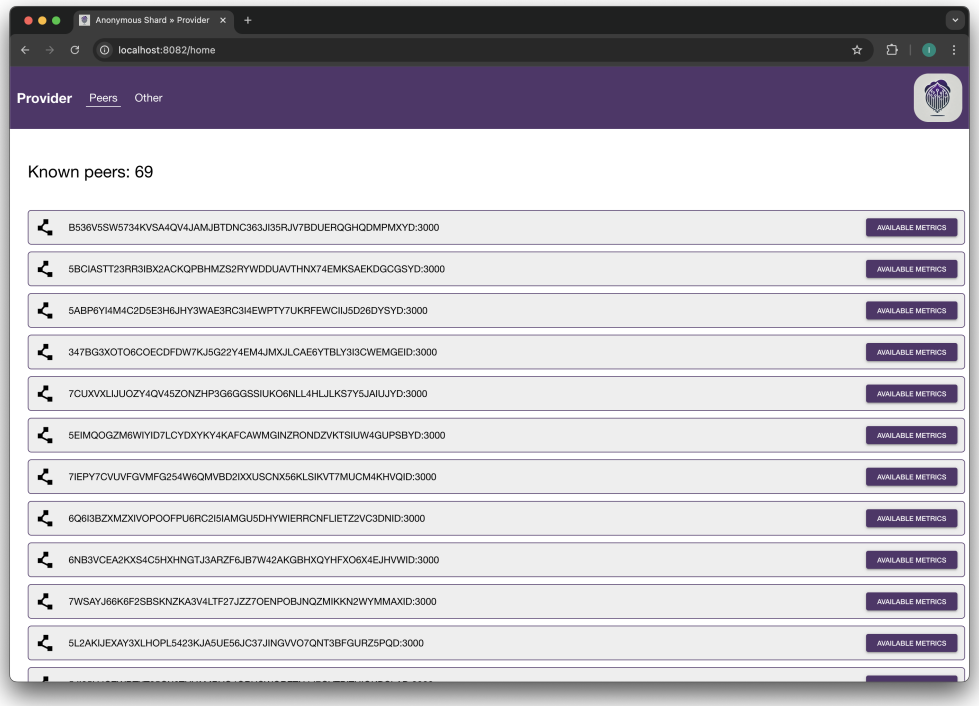

Figure 5.2: Bucket Size 70 saturated

5.2 Average RTT Test

A series of tests were conducted to measure the average Round Trip Time between nodes in the network with the final goal of estimating possible message delivery time values when dealing with a real-world scenario. RTT is a critical metric for evaluating system performance, as it directly impacts on message delivery and so on task execution time. For this test, the specially created *Analyzer* prototype was used.

5.2.1 Process

In this experiment, *Entry* and *Exit* nodes were used to measure the time taken for a message to traverse the Tor network. The Analyzer sends a message to the Entry node, which then forwards it to the Exit node. The total time taken for the message to reach its destination and return is recorded. A recap on what are Entry and Exit nodes is provided in the Design and Implementation Chapter 4.7.2. To get a quite accurate average RTT value,

a set of 50 requests was repeated each hour of the day with a total of 1200 requests per configuration. The number of requests is a tunable parameter that can be increased to have a more accurate average value if needed or if the experiment needs to be repeated for any reason.

At the end of the process, the measurements were averaged per hour, and charts were generated to visualize the results clearly and provide a comprehensive view. Moreover, the standard deviation was calculated to understand the variance in RTT values and identify potential outliers or noteworthy time spans.

Considerations

Before analyzing the results or taking a look at the configurations, it is essential to consider how the measurements are obtained and the factors that can influence them. The tests have been conducted using a Dockerized environment, where two containers simulated the client and the server respectively.

To better explain and understand the internal mechanisms, an example is examined. In the Russia-USA case, the client was explicitly configured to use an entry node in Russia (`EntryNodes ru`) and an exit node in the United States (`ExitNodes us`). The server was only configured to use an exit node in the United States (`ExitNodes us`). This setup ensured that all traffic was routed through the specified locations.

Although both the client and the server have been hosted on a machine in Italy, which may introduce slight bias in the results. This setup effectively simulates a controlled observer-based measurement; however, it does not fully replicate a globally distributed scenario. Still, it offers valuable insights into Tor network performance across different geographic configurations.

The experiment was repeated with different entry-exit node pairs across the various countries configurations.

The results revealed significant differences in RTT depending on the geographic distance between the nodes and the specific countries involved as expected. Circuits spanning longer distances generally exhibited higher latency, while those confined to geographically closer nodes showed lower RTT values. Time-based variations also highlighted the impact of network congestion, with peak latency periods aligning with high-traffic hours in certain regions.

Specific Measurement Process

The total RTT consists of the time taken for a message to travel from the client to the server through the Tor network and back, but as mentioned, the client and the server were hosted in Italy. So, the RTT is not a simple client-server ping but involves a multistep process.

The measurement, keeping the example of the USA-Russia configuration, can be divided into three main phases:

Request: Steps involved from the client to reach the server through the Tor network.

1. **Client to Entry Node:** Italy → RU.
2. **Entry Node to Relay Nodes:** RU → Relays.
3. **Relay Nodes to Exit Node:** Relays → US.
4. **Exit Node forwards to Server:** US → Italy.

Response: In this phase, the response is sent back to the client via the same Tor path.

1. **Server forward through Exit Node:** Italy → US.
2. **Exit Node to Relay Nodes:** US → Relays.
3. **Relay Nodes to Entry Node:** Relays → RU.
4. **Entry Node to Client:** RU → Italy.

Note that Tor circuit routing time, relay selection, and forwarding are included in the measurements. Same for the relay processing delay and network propagation delay between relays.

Other Latency Factors: Several other elements impact RTT measurements.

1. **Tor Network Congestion**: Congestion within the Tor network can introduce additional delays, particularly when relays become overloaded due to high traffic volume at specific times or on certain routes. Another contributing factor is the limited bandwidth of exit nodes, which can further slow down data transmission when handling large amounts of traffic.

2. **Operating System and Docker Overhead**: Although not directly related to the Tor network, system-level factors such as Docker container networking overhead and OS-related processing delays can contribute to additional latency. While these factors are generally minor, they should still be considered in performance evaluations.

Note that the Tor Circuit Setup Time is not included in the RTT measurements, as it is a separate factor that can vary depending also on relay availability. This setup time can be a significant factor in countries where the Tor network is either unavailable or rarely used, and some considerations about this phase can be found in the Encountered Problem Section 5.2.4.

Final RTT Computation: The total is the sum of all contributing factors.

$$T_{Request} = (T_{\text{Client}\to\text{Entry}} + T_{\text{Entry}\to\text{Relays}} + T_{\text{Relays}\to\text{Exit}} + T_{\text{Exit}\to\text{Server}})$$
$$T_{Response} = (T_{\text{Server}\to\text{Exit}} + T_{\text{Exit}\to\text{Relays}} + T_{\text{Relays}\to\text{Entry}} + T_{\text{Entry}\to\text{Client}})$$
$$RTT = T_{Request} + T_{Response}$$

Where each term includes network latency, encryption/decryption overhead, relay processing time, and congestion delays.

5.2.2 Countries Configuration

These tests were conducted using different country configurations trying to evaluate the system's message delivery time across various geographic locations.

1. **Italy-Italy:** This configuration serves as a baseline to understand the behavior of the system in a controlled environment. By testing within the same country, we can minimize the effects of network congestion

and routing complexity, providing a reference to comparing other, more geographically spread configurations. In the following sections, this configuration will be called the basecase.

2. **Italy-Bulgaria:** The Italy-Bulgaria pair helps us analyze the impact of geographical distance and network congestion in the case of a longer, but still European, distance. This test will highlight how the system performs when routing through different countries within Europe, and how the performance changes as moving away from the baseline (Italy-Italy) configuration. What should be noted are timings similar to the basecase with a slight increase in the values due to the increased distance between the two countries.

3. **Italy-Mexico:** By introducing a significant transcontinental distance, the Italy-Mexico configuration will allow us to evaluate the system's performance under different routing conditions. This test explores how the system behaves when traversing multiple international networks with potential bandwidth limitations and routing inefficiencies.

4. **Russia-USA:** A test spanning the continents of Europe and North America will evaluate the system's performance over long distances. The results will help assess the impact of both physical distance and network congestion, which could vary due to differing levels of infrastructure and routing mechanisms across countries.

5. **USA-India:** This configuration tests a long-distance route between North America and Asia, which often experiences significant network congestion due to the amount of traffic routed through these regions. The USA-India test will offer insights into how well the system handles latency and congestion in the context of transoceanic connections, often influenced by both infrastructural limitations and geopolitical factors.

6. **Japan-France:** A test between Japan and France evaluates intercontinental network performance between Asia and Europe. This route often involves different network backbones and potential bottlenecks. This configuration will help understand how the system copes with varying network performance influenced by different regional routing schemes.

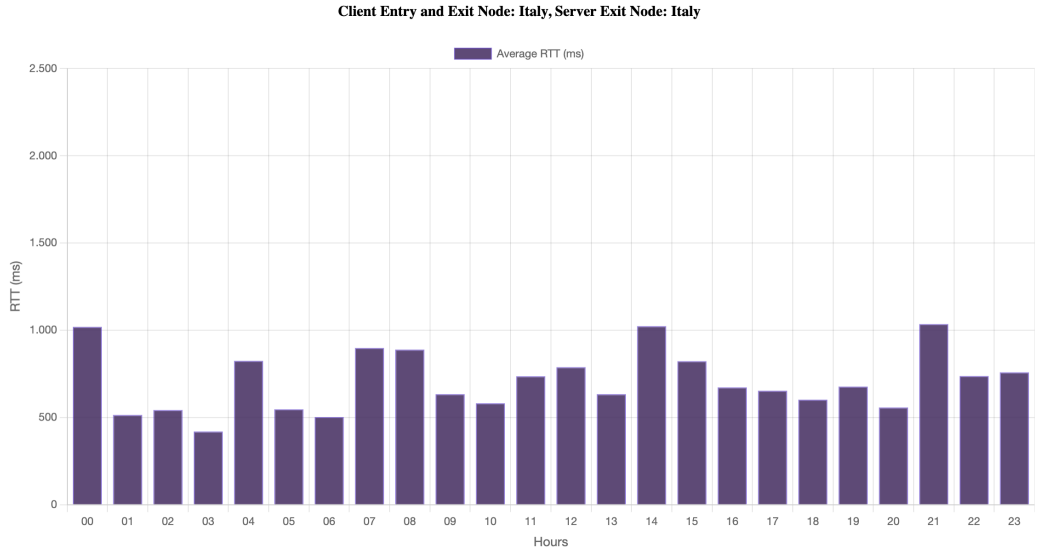

Figure 5.3: AVG Italy-Italy

Each tested country configuration is represented in the figures below in the Detailed Analysis Section 5.2.3, where the average RTT for each location is displayed.

5.2.3 Detailed analysis

This section presents an analysis of the results obtained following the methodology described earlier. As introduced, a chart for each country configuration is provided to offer a comprehensive view of network latency.

The analysis begins with the baseline configuration and gradually extends to other setups, depending on the geographic distance covered by the tests. The baseline configuration refers to the scenario where the client, server, and both the entry and exit nodes are located in Italy. This setup serves also as a reference for comparison, as it represents the minimum latency in these tests.

Figure 5.3 illustrates the average RTT for the Italy-Italy configuration.

From the chart in Figure 5.3, it can be observed that the average RTT over time is approximately 710 ms, with a standard deviation of 170 ms. The relatively low standard deviation indicates that the network remains fairly stable throughout the day. However, as expected, there is a slight increase in RTT during the morning and evening, while the lowest latency is recorded

at night. The impact of the Tor network on RTT is clear from this baseline case.

A more interesting comparison emerges in the Italy-Bulgaria configuration, as shown in Figure 5.2.3. In this scenario, as described in Section 5.2.2, the physical distance between the nodes increases. Consequently, the average RTT rises to approximately 760 ms, with a standard deviation of 245 ms.

This result is noteworthy because, while an increase in RTT was expected due to the longer distance, the difference remains moderate, as both countries are within Europe. Additionally, the standard deviation grows slightly more than the RTT itself. This observation suggests that as the geographical distance increases and additional network routes are involved, not only RTT rises, but the variability in network performance also increases. The growing standard deviation indicates a degree of network instability over time, but it still remains within acceptable limits.

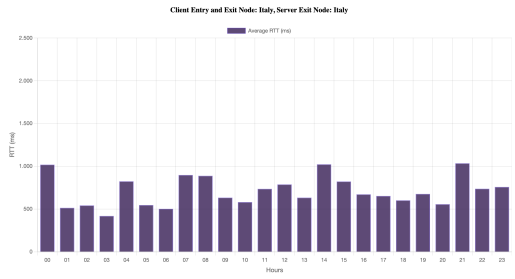

Figure 5.4: AVG Italy—Italy

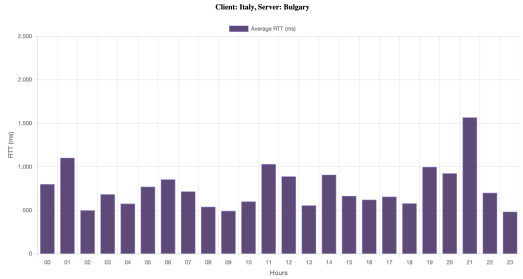

Figure 5.5: AVG Italy—Bulgary

An even more significant variation is observed in the Italy-Mexico configuration in Figure 5.6. In this case, the distance increases considerably, and the network infrastructure differs more significantly. The average RTT in this setup is approximately 815 ms, with a standard deviation of about 300 ms. As expected, both RTT and standard deviation increase with distance, confirming the correlation between geographical separation and network performance variability.

Examining all three charts, it becomes clear that between 05:00 PM and 07:00 PM, the RTT values are higher. This pattern, unlike the European or intra-Italy measurements and averages, represents the peak hours during which network congestion is more pronounced.

The Russia-USA configuration, shown in Figure 5.7, exhibits different results compared to the previous configurations.

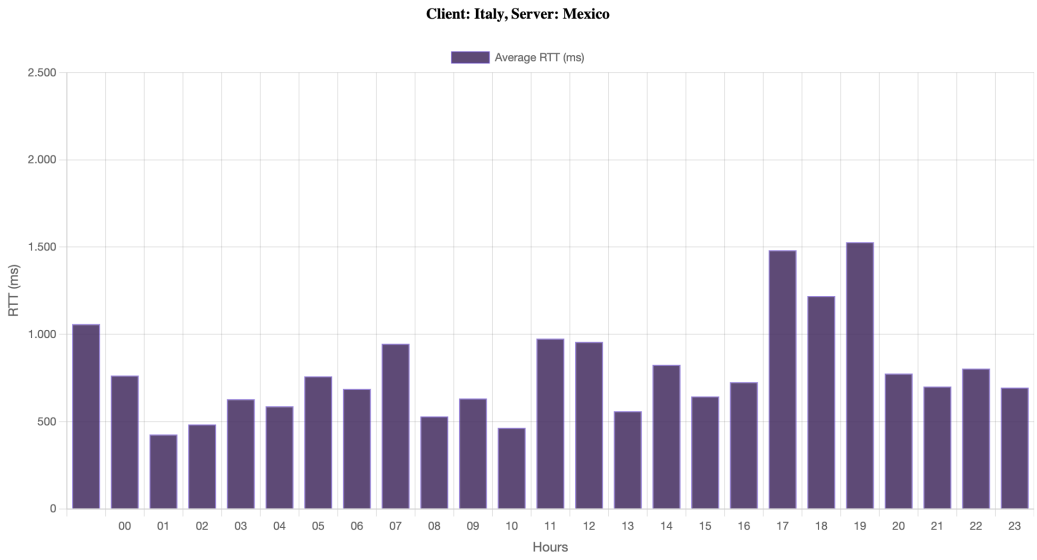

Figure 5.6: AVG Italy-Mexico

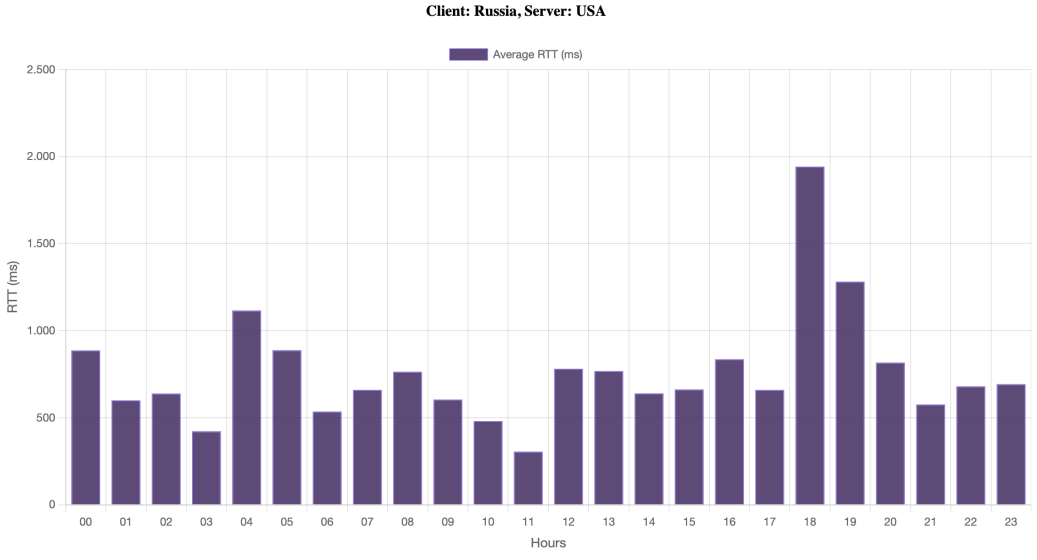

Figure 5.7: AVG Russia-USA

An interesting aspect of this configuration is that the average RTT is approximately 820 ms, with a standard deviation of 320 ms. More than the increase in RTT, which remains consistent with the expected latency increase due to the greater distance between the nodes, what stands out is its similarity to the Italy-Mexico case. This similarity may be attributed to various factors, particularly the choice of the circuit. Since Tor selects nodes based only on the specified country, the actual nodes chosen in Russia and the USA significantly impact the results. A possible explanation is that the Tor network does not select nodes randomly within a country but rather uses an algorithm that balances multiple parameters, such as latency, network capacity, and load. In this case, despite the greater geographical distance, the RTT values are similar to those in the Italy-Mexico configuration, highlighting the crucial role of circuit selection. Additionally, this result suggests that, due to the high availability of nodes in the USA, the network includes significantly more high-performance nodes compared to Mexico. This is clear from the current number of available relays: 9 in Mexico of which only one Entry (or Guard) and one Exit versus the USA where there are a total of almost 1700 relays[1]. Tor does not randomly choose nodes within a country but instead optimizes the selection based on network conditions. As a result, the bandwidth and latency of the nodes in the USA are generally better than those in Mexico, leading to similar RTT values despite the greater distance. This result is particularly relevant as it demonstrates how circuit selection can significantly influence system performance. Furthermore, an observation is that a peak in RTT was recorded, not during a specific time window, but rather at 09:00 PM, causing RTT spikes of nearly 2000 ms.

The USA-India configuration, shown in Figure 5.8, shows an even more pronounced increase in RTT compared to the previous configurations. In this case, the measurements aim to analyze how the network handles transoceanic traffic routing and to assess whether infrastructural limitations or other factors influence performance. The average RTT is approximately 850 ms, with a standard deviation of 440 ms.

A notable aspect of this configuration is the significant peak in RTT values, which exceed 2500 ms around 06:00 PM. The difference between this time period and other hours is significant. Further investigation could help determine whether this peak is an isolated event or a recurring pattern. Conducting measurements over weeks or months would provide insights into

[1]According to Tor Metrics website at the time of testing

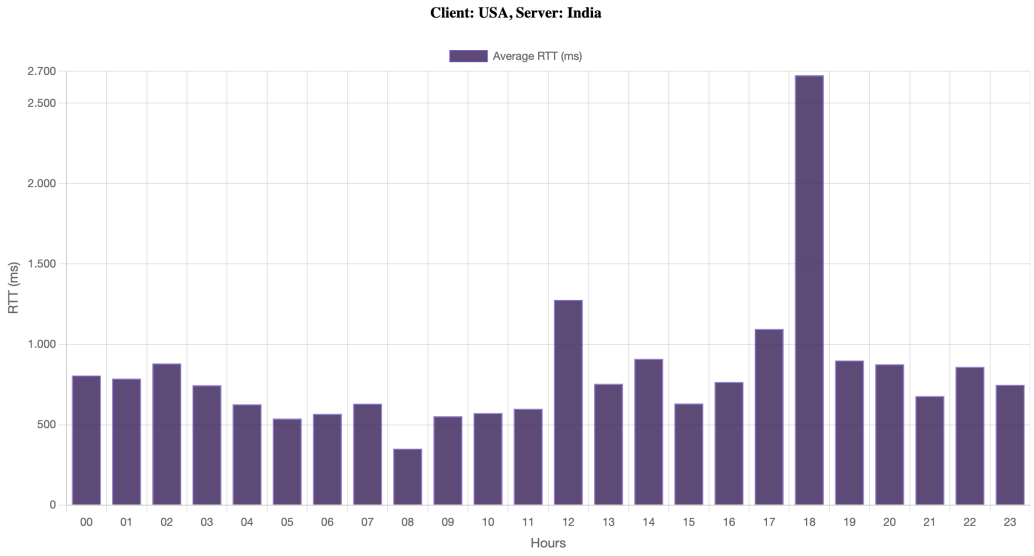

Figure 5.8: AVG USA-India

whether this behavior is consistent or specific to that particular day.

Aside from this peak, the measurements are generally consistent with the expected increase in RTT due to the increased distance between the nodes. However, the high standard deviation, over 400 ms, indicates significant variability in network performance. While this value is influenced by the extreme peak observed, it remains a noteworthy finding in itself.

The measurement about the Japan-France configuration has been done to evaluate the system over intercontinental network performance between Europe and Asia. The average RTT is approximately 740 ms, with a standard deviation of 290 ms, as shown in Figure 5.9. The results are consistent and no bottlenecks, but three possible peaks that reached the maximum of about 1500 ms are noted. They are isolated, so they can be considered situations in which the network was a little bit more congested but, like in the USA-India configurations, further investigation could help determine whether these peaks are recurring patterns or not.

In summary, these tests provide valuable insights into the system's performance across different geographic configurations. The results highlight the impact of physical distance, network congestion, and circuit selection on network latency. Moreover, the standard deviation values offer insights into network stability and variability over time.

The prototype is available in this GitHub repository, and the configu-

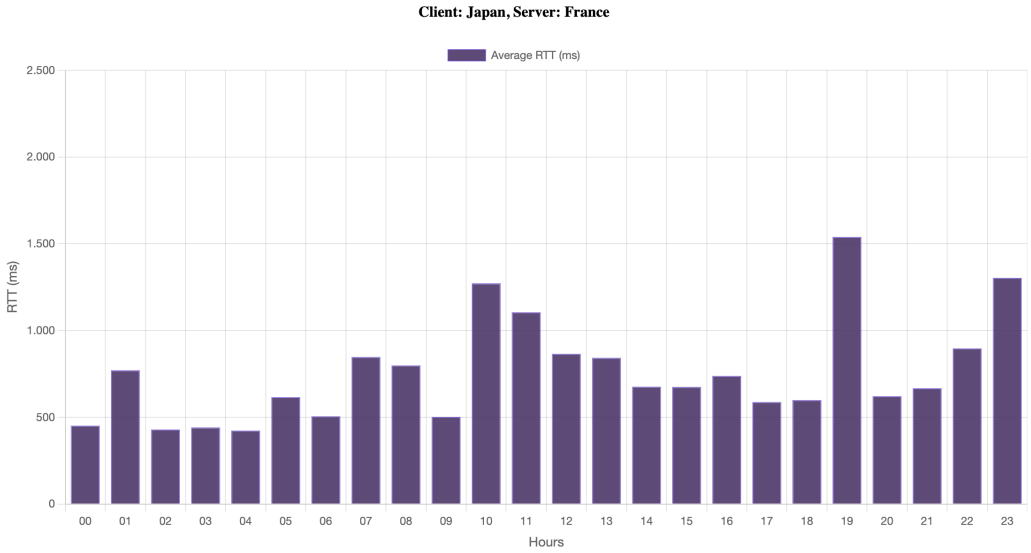

Figure 5.9: AVG Japan-France

ration file can be easily modified to include new countries or to change the entry and exit nodes. A list of available relays available in different countries and interesting metrics can be found in the Tor Metrics website.

5.2.4 Encountered problems

One of the first issues encountered during this phase was the time required for the Tor network's bootstrap process. Initially, I assumed this was influenced by the choice of entry or exit nodes, as the Tor network could also have no running relays in the specified country. So, selecting an entry or exit node in such locations could lead to long bootstrap times, and in some cases, if the *UseStrictNodes 1* line is present in the *torrc* configuration file [2], the bootstrap process could fail. This occurred when there were no suitable entry or exit nodes available in the specified region, preventing the bootstrap process from completing.

Another observed issue was the variation in bootstrap performance depending on the network environment. For instance, when connecting from my home network, the process is completed quickly with valid entry and exit nodes. However, when using the University network, the bootstrap process was noticeably slower. A possible explanation for this discrepancy is that

[2]This option force Tor to use Entry or Exit node if set to 1, failing if no node is available

different ISPs have varying agreements with networks that route Tor traffic or more specifically a firewall rule that tries to perform Deep Packet Inspection (DIP) slowing down the entire process. Additionally, network policies may prioritize certain types of traffic over others.

A potential solution to mitigate these issues is the use of **Tor Bridges** to bypass network-level throttling or censorship. Tor bridges are private, unlisted entry points into the Tor network that help circumvent restrictions imposed by ISPs.

5.2.5 Limitations

The limitations of this testing phase are primarily related to the controlled environment in which the tests were conducted. The setup did not present some issues, in the sense that the tests were successfully completed. The primary limitation is that the tests were conducted in a local environment deploying the client and server on the same machine. Measuring the round-trip time (RTT) between nodes in a real-world scenario, where nodes should be distributed globally, would provide more accurate and relevant results. Additionally, in a real scenario, the locations of peers are not known, meaning the RTT can vary significantly depending on the physical distance between nodes. The tests were conducted in a controlled environment, and the results may not accurately reflect the exact system performance in a real-world setting but can give an idea of how the system can behave with different configurations. Moreover, since the Tor network is a shared network, the results are influenced by other users' activities and overall network congestion.

The task execution could be subject to redirection before execution and completion, introducing additional delays that depend on the physical location of the new selected provider. This limitation can be addressed in future tests by deploying instances worldwide to evaluate system performance in a real-world setting, and by completing the Kademlia full discovery algorithm creating a way to represent coherently the distance between nodes. Other improvements to mitigate these limitations are discussed in the *Future Test Extensions* Section 5.3.

Furthermore, to get significant results and generate charts representing the average RTT over time, the tests should be conducted over an extended period (e.g., several months) to calculate a more accurate average RTT over

really worldwide distributed nodes. In particular, a subset of nodes could execute these tests continuously to provide a more comprehensive view of network latency and performance at each task submission.

An additional factor that could affect the measurements in a real-world scenario is the limited number of Tor nodes in certain countries. In this case, the bootstrap process could fail, as discussed in the *Encountered Problems* section 5.2.4.

5.3 Future Test Extensions

To improve the testing phase and get more comprehensive results, some extensions to the current methodology can be considered. Moreover, the limitations encountered during the tests can be addressed by implementing additional features and conducting more extensive evaluations. Additionally, metrics such as bandwidth and available relays can be considered to assess the system's performance. Measuring other aspects of the network, such as Circuit Setup Time, can also provide insights into network behavior under different configurations.

Beyond this approach, other improvements that can be explored:

1. **Mocking:** Mocking particular components of the system to simulate different conditions and evaluate system behavior under various scenarios. This approach could be a valuable tool for testing edge cases and assessing system robustness or to probe specific behaviors for feasibility checks.

2. **Real-World Scenario:** Setting up instances worldwide exploiting cloud services to analyze the system's performance in a truly distributed setting, allowing for realistic measurements and better insights.

3. **Network:** Evaluating the system's performance in various configurations, including fully connected networks, trying hierarchical structures (maybe more to adapt the system for other uses), and other sparse peer-to-peer topologies.

4. **Robustness:** Assessing the system's fault tolerance by simulating node failures, abrupt disconnections, and high-load scenarios. In this

case, retry mechanisms and fault tolerance mechanisms have been tested manually to ensure that the system can recover from errors and continue functioning correctly, but a more structured approach is for sure beneficial.

5. **Security:** Simulating network attacks, such as Sybil Attacks [10] and traffic correlation attempts, to evaluate the resilience of the system against potential threats.

6. **Optimization:** Testing and comparing different routing mechanisms to determine the most efficient strategy for task distribution and peer discovery, including the full Kademlia discovery algorithm.

By incorporating these extensions, the testing phase can be significantly improved, leading to more meaningful performance evaluations.

Conclusions

Concluding this thesis project has been a challenging and rewarding experience. Since it was a project that I started from scratch and with no clear boundaries, difficulties and challenges to face were numerous, but the satisfaction of having a prototype is huge. At the beginning of the project, the difficulties were mainly non-technical, such as defining the scope of the project and the objectives to be achieved. When a good trade-off between basic abstraction and complexity was found, the idea of the project was clear.

After the initial phase, the technical challenges were numerous. My initial intention was to focus on high-level solutions, leveraging libraries like *Libp2p* to build the network layer and concentrate on the business logic. However, the issues encountered with the network layer were more complex as explained in the Transport Layer Section 4.7.3 and some goals could not be achieved with certainty. As a result, trade-off were made to ensure the project's completion within the constraints by deciding to build a simple yet effective network management layer to allow nodes to communicate. Although this approach was more time-consuming and not much time was spent on expanding the business logic, I believe the goal of spreading the idea that an innovative and anonymous ecosystem is necessary was achieved.

What is needed in this age is a globally recognized and widely used system that ensures privacy and anonymity, while maintaining a high level of decentralization and censorship resistance. The concept behind this prototype is to build towards such an ecosystem, where users own computational resources and can use them to perform tasks in a secure and anonymous environment. The ultimate goal is to create a stable ecosystem that can be accessed and used by anyone, without the need for a central authority overseeing the system, which is a significant strength.

This prototype serves as a foundation for expanding this concept or creating something new with the same underlying principles. The next Future

Directions Section will further explore this ecosystem concept and analyze potential future extensions of the project.

Additionally, the second prototype presents a strong concept: introducing new nodes into the system to support its overall functionality. While the results are currently analyzed manually, envisioning an automated mechanism to monitor and analyze the network status could be a very nice feature.

Future Extensions

In this section, possible future extensions of the project are analyzed. Since this prototype is just the beginning of a larger vision for an anonymous computing ecosystem, the possibilities for future improvements are huge.

For first, enhance the network layer by leveraging tested and reliable libraries, such as *Libp2p*, while refining the business logic.

If, for any reason, contributing to these libraries proves to be unsuccessful, the fallback plan is to build a custom network layer starting from the developed one for this prototype. It would be costly but very effective.

In addition to improving the network layer, expanding the business logic is essential. There are many opportunities for improvement in this regard, beginning with the expansion of the core node types and the introduction of new node typologies. The system is designed to be modular, enabling easy adaptations to meet new possible specific requirements.

If the need for task executors with specialized functionality arises, new node types could be created to handle those tasks. This flexibility ensures the system's adaptability and scalability. Furthermore, building a small but active community could greatly enhance the growth of such a system. Open-source executors or evaluators packages could be released based on the community's needs and requests. Moreover, for example, developers can create custom node types with their own business logic and integrate them into the system.

The system could be enhanced by incorporating advanced cryptographic techniques, such as Homomorphic Encryption for some operations and Trusted Execution Environments (TEEs) to ensure the integrity of the system. In general, privacy-enhancing technologies can be evaluated to improve the system's security and privacy on different levels.

Another important aspect to consider is the sustainability and attractiveness of the system to users, and for this purpose, a reward system could

be introduced. This system could be implemented using Monero or Z-Cash as a cryptocurrency to ensure and keep high anonymity level. From an initial analysis, Monero could be a better choice, as it does not allow public transactions, while Z-Cash permits the choice between public and private. This mechanism not only permits the system to be self-sustainable but also incentivizes users to participate and contribute to the network. Moreover, it allows thinking beyond the simple reward for task execution or processing. In particular, new types of nodes, designed solely to support the network, could be introduced. These supporter nodes would help maintain system consistency and ensure its continuous operation. While the rewards for these nodes would be lower than those for processing nodes, their resource requirements would also be less demanding, making them more accessible to a broader range of participants.

A reconsideration of the peer selection process and the task redirection mechanism is also necessary. To optimize task routing, a more efficient routing mechanism could be implemented, possibly leveraging new metrics provided by the Tor network or by supporters. This would allow tasks to be directed to the most optimal nodes, gaining clear improvements in the system's performances. In this case, a task classification mechanism could be introduced to weigh tasks based on their complexity and importance, introducing the concept of task priority also possibly linked to the reward system. More metrics regarding the nodes could be provided to identify malicious nodes or sparsity in the network trying to better address the network status and the nodes' decentralization level.

An alternative approach, for which the software is not designed, could be to introduce de-anonymization mechanisms. This is the case in which the system is used for other purposes, and the involved parties need a mechanism to de-anonymize results in case of disputes or other reasons. The system should be revisited, and some secret sharing, threshold cryptography or similar techniques could be introduced to achieve this result. These techniques involve trusted parties that can de-anonymize the results, but only if a certain number of parties agree. While this is not something I would like to see in the system, it is a possibility that should be considered for other declinations.

What I believe it could be interesting to integrate the system with a Decentralized Autonomous Organization (DAO), where participants can vote on the future directions. This integration could be interesting, but it requires

a deeper analysis, as it could also present potential risks, making it a double-edged sword keeping in mind the more complex is the system, the more vulnerabilities it could have. In this case, not only in the technical sense but also in the governance one.

Finally, after these considerations, I would like to underline that continuous research and development are essential for the growth and sustainability of any technology, enabling people to adapt to emerging challenges and stay safe. Through R&D and innovative solutions, it should be fundamental to ensure that systems can operate effectively in restrictive environments, such as censored countries or other challenging circumstances. I'm not sure if this system will help anyone, but I'm really proud to have thought about this topic and contributed today, as I will in my future.

Bibliography

[1] Safwan Mahmud Khan and Kevin W. Hamlen. AnonymousCloud: A Data Ownership Privacy Provider Framework in Cloud Computing. In *2012 IEEE 11th International Conference on Trust, Security and Privacy in Computing and Communications (TrustCom)*, pages 170–176, 2012. IEEE. 10.1109/TrustCom.2012.94. Url: `https://doi.org/10.1109/TrustCom.2012.94`

[2] Len Bass, Paul Clements, and Rick Kazman. Quality Attribute Scenarios and Tactics. In *Software Architecture in Practice*, 3rd edition, chapter 4, pages 69–94. Addison-Wesley, 2012. ISBN 978-0-321-81573-6.

[3] Petar Maymounkov and David Mazieres. Kademlia: A Peer-to-Peer Information System Based on the XOR Metric. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 53–65, 2002. Springer, Cambridge, MA, USA. `https://dl.acm.org/doi/10.5555/646334.687801`.

[4] Martin Fowler. Ubiquitous Language. 2006. `https://martinfowler.com/bliki/UbiquitousLanguage.html`.

[5] Alistair Cockburn. *Writing Effective Use Cases*. Addison-Wesley Professional, 2000. ISBN 978-0201702255.

[6] Robert C. Martin. *Clean Architecture: A Craftsman's Guide to Software Structure and Design*. Prentice Hall, 2017. ISBN 978-0134494166.

[7] Robert C. Martin. The Dependency Inversion Principle. *C++ Report*, 8(6):61–66, 1996. `https://web.archive.org/web/20110723210920/http://www.objectmentor.com/resources/articles/dip.pdf`.

[8] David Garlan, Robert Allen, and John Ockerbloom. Architectural Mismatch: Why Reuse Is So Hard. In *Proceedings of the 17th International Conference on Software Engineering (ICSE)*, pages 179–185, 1995. ACM. 10.1145/225014.225031. Url: https://dl.acm.org/doi/10.1145/225014.225031

[9] Dimitris Vyzovitis, Yusef Napora, Dirk McCormick, David Dias, and Yiannis Psaras. GossipSub: Attack-Resilient Message Propagation in the Filecoin and ETH2.0 Networks. *arXiv preprint arXiv:2007.02754*, 2020. https://arxiv.org/abs/2007.02754.

[10] John R. Douceur. The Sybil Attack. In *Proceedings of the 1st International Workshop on Peer-to-Peer Systems (IPTPS)*, pages 251–260, 2002. Springer. ISBN: 978-3-540-45748-0

[11] Kathleen A. Wallace. Anonymity. *Ethics and Information Technology*, 1(1):23–35, 1999. Kluwer Academic Publishers. 10.1023/A:1010076217393. Url: https://doi.org/10.1023/A:1010066509278

[12] Timothy C. May. The Crypto Anarchist Manifesto. https://nakamotoinstitute.org/authors/timothy-c-may/.

[13] David Chaum. Untraceable Electronic Mail, Return Addresses, and Digital Pseudonyms. *Communications of the ACM*, 24(2):84–90, 1981. ACM. 10.1145/358549.358563. Url: https://doi.org/10.1145/358549.358563

[14] Daniel J. Solove. 'I've Got Nothing to Hide' and Other Misunderstandings of Privacy. *San Diego Law Review*, 44:745–772, 2007. https://ssrn.com/abstract=998565.

[15] Helen Nissenbaum. *Privacy in Context: Technology, Policy, and the Integrity of Social Life*. Stanford University Press, Redwood City, 2009. ISBN 9780804772891. https://doi.org/10.1515/9780804772891. Url: https://doi.org/10.1515/9780804772891

[16] Commission services. End-to-end encryption in criminal investigations and prosecution. *Note from the Commission services*, September 18, 2020. Brussels. https://data.consilium.europa.eu/doc/document/ST-10730-2020-INIT/en/pdf.

[17] Albert Teich, Mark S. Frankel, Rob Kling, and Ya-Ching Lee. Anonymous Communication Policies for the Internet: Results and Recommendations of the AAAS Conference. *The Information Society*, 15(2):71–77, 1999. Taylor & Francis. Url: https://doi.org/10.1080/019722499128538.

[18] Leon Mann. The Baiting Crowd in Episodes of Threatened Suicide. *Journal of Personality and Social Psychology*, 41(4):703–709, 1981. American Psychological Association. Url: https://doi.org/10.1037//0022-3514.41.4.703.

[19] John Suler. The Online Disinhibition Effect. *Cyberpsychology & Behavior*, 7(3):321–326, 2004. Mary Ann Liebert, Inc. Url: http://dx.doi.org/10.1089/1094931041291295

[20] Kimberly M. Christopherson. The Positive and Negative Implications of Anonymity in Internet Social Interactions: 'On the Internet, Nobody Knows You're a Dog'. *Computers in Human Behavior*, 23(6):3038–3056, 2007. Elsevier. Url: http://dx.doi.org/10.1016/j.chb.2006.09.001

[21] Eric Hughes. A Cypherpunk's Manifesto. 1993. https://www.activism.net/cypherpunk/manifesto.html. Accessed: 2025-02-20.

[22] Timothy C. May, "The Cyphernomicon: Cypherpunks FAQ and More," 1994. [Online document]. Available: https://cdn.nakamotoinstitute.org/docs/cyphernomicon.txt.

[23] Carole Cadwalladr and Emma Graham-Harrison. Revealed: 50 million Facebook profiles harvested for Cambridge Analytica in major data breach. *The Guardian*, March 17, 2018. Url: https://www.theguardian.com/news/2018/mar/17/cambridge-analytica-facebook-influence-us-election.

[24] David Chaum, "Security Without Identification: Transaction Systems to Make Big Brother Obsolete," *Communications of the ACM*, vol. 28, no. 10, pp. 1030–1044, 1985, doi: 10.1145/4372.4373. Url: https://doi.org/10.1145/4372.4373

[25] Julian Assange, Jacob Appelbaum, Andy Müller-Maguhn, and Jérémie Zimmermann, *Cypherpunks: Freedom and the Future of the Internet*, New York: OR Books, 2012, ISBN 978-1-939293-00-8.

[26] Paul Ohm. Broken promises of privacy: Responding to the surprising failure of anonymization. *UCLA Law Review*, 57:1701–1777, 2010. `https://ssrn.com/abstract=1450006`.

[27] Edward Snowden. *Permanent Record*. Metropolitan Books, 2019. ISBN 978-1-250-23123-8.

[28] Andrew S. Grove. *Only the Paranoid Survive: How to Exploit the Crisis Points That Challenge Every Company and Career*. Currency Doubleday, 1996. ISBN 978-0-385-48258-2.

[29] United Nations General Assembly. Resolution 68/167: The right to privacy in the digital age. *United Nations*, 2013. `https://docs.un.org/en/A/RES/68/167`.