WILLIAM TAKESHI PEREIRA

SCALABILITY ANALYSIS OF AN IOT PLATFORM: THE SWARMOS SCALABILITY

Corrected Version

São Paulo
2023

WILLIAM TAKESHI PEREIRA

SCALABILITY ANALYSIS OF AN IOT PLATFORM: THE SWARMOS SCALABILITY

Corrected Version

Dissertation presented to the Polytechnic School of University of São Paulo to obtain a Master's Degree in Electrical Engineering.

São Paulo
2023

WILLIAM TAKESHI PEREIRA

SCALABILITY ANALYSIS OF AN IOT PLATFORM: THE SWARMOS SCALABILITY

Corrected Version

Dissertation presented to the Polytechnic School of University of São Paulo to obtain a Master's Degree in Electrical Engineering.

Program:

Electronic Systems

Advisor:

Laisa Caroline Costa de Biase

São Paulo
2023

Este exemplar foi revisado e corrigido em relação à versão original, sob responsabilidade única do autor e com a anuência de seu orientador.

São Paulo, _____ de _____ de _____

Assinatura do autor: _____

Assinatura do orientador: _____

Catalogação-na-publicação

Pereira, William Takeshi
Scalability Analysis of an IoT Platform: the SwarmOS Scalability / W. T. Pereira -- versão corr. -- São Paulo, 2024.
130 p.

Dissertação (Mestrado) - Escola Politécnica da Universidade de São Paulo. Departamento de Engenharia de Sistemas Eletrônicos.

1.Internet das Coisas 2.Escalabilidade 3.Testes 4.Swarm 5.Sistemas Distribuídos I.Universidade de São Paulo. Escola Politécnica. Departamento de Engenharia de Sistemas Eletrônicos II.t.

Aos meus pais e meu irmão.

ACKNOWLEDGMENTS

Many are the persons who deserve my gratitude for helping me during the development of this work.

I would like to begin by acknowledging my parents, Geraldo E. Pereira, Rozana T. S. Pereira. Despite my career choice leading to involvement in areas unfamiliar to them, they have always been a source of encouragement, inspiring me to pursue my chosen path with determination and enthusiasm. And my brother Victor T. Pereira, whose support and camaraderie have been constants in my life.

I would like to express my sincere gratitude to my advisors, Laisa C. C. De Biase and Marcelo K. Zuffo, who have provided valuable guidance and support throughout my project. Their expert knowledge and experience have been instrumental in shaping my research and helping me navigate the complexities of the field. I am deeply thankful for their unwavering support and encouragement, which has been an essential factor in my academic and personal growth. I appreciate the time and energy they have dedicated to my success.

I would like to extend my heartfelt thanks to my colleague Geovane Fedrecheski for their invaluable support and collaboration throughout this project. His input and contributions have been invaluable, and I have learned a great deal from working alongside them. I appreciate their collaborative spirit, positive attitude, and willingness to lend a helping hand.

Also, my acknowledgements to the Swarm Team and all who I have interacted with at Universidade de São Paulo. I am honored to have had the opportunity to work with such knowledgeable and talented individuals. I will always be grateful for their contributions to my education and my future.

Finally, my acknowledgments to USP, who gave me all infrastructure and support for me to develop this work.

RESUMO

Esta pesquisa adentra no vasto cenário da Internet das Coisas (IoT) através da inovadora perspectiva da computação em enxame, inspirada em sistemas biológicos. Sistemas de enxame, compostos por agentes simples, exibem a capacidade de autoorganização e abordam colaborativamente desafios complexos por meio de interações. O ponto central é o desafio inerente de escalabilidade na IoT, onde conectar inúmeros dispositivos se torna uma empreitada complexa. Este estudo visa definir e aprimorar a escalabilidade de sistemas de enxame. Para isso, o trabalho introduz um robusto ambiente de teste virtualizado projetado para explorar técnicas de escalabilidade. Atuando como um passo intermediário entre a concepção e testes em dispositivos físicos, este ambiente proporciona uma abordagem ágil e flexível. A ênfase reside não apenas em conceituar e aprimorar um ambiente de teste rigoroso, mas também em aprimorar um sistema de IoT, especificamente o SwarmOS, com um foco dedicado na escalabilidade, abordando desafios críticos no amplo domínio da IoT.

Palavras-Chave – Internet das Coisas, Escalabilidade, Testes, Swarm, Sistemas Distribuídos.

ABSTRACT

This research explores the vast landscape of the Internet of Things (IoT) through the lens of swarm computing, an innovative approach inspired by biological systems. Swarm systems consist of simple agents that, through self-organization, collaboratively address complex challenges via interactions. The inherent scalability challenge in IoT, connecting a myriad of devices, becomes a focal point. The study seeks to define and improve the scalability of swarm systems. For this, this work introduces a robust virtualized testbed to probe scalability techniques, this testbed introduces a lean and flexible environment to be a intermediate step between the idea and testing in physical devices. The emphasis lies not only in conceptualizing and refining a rigorous testing environment but also in enhancing an IoT system, specifically the SwarmOS, with a dedicated focus on scalability addressing critical challenges in the expansive realm of the IoT.

Keywords – Internet of Things, Scalability, Testbed, Swarm, Distributed Systems, Virtualization.

LIST OF FIGURES

LIST OF TABLES

ACRONYMS

ABAC Attribute-Based Access Control

ACL Access Control Lists

DHT Distributed Hash Table

DID Decentralized Identifier

HGABAC Hierarchical Group and Attribute-Based Access Control

ICE Interactive Connectivity Establishment

IGDP UPnP Internet Gateway Device Protocol

IoT Internet of Things

IPv4 Internet Protocol version 4

IPv6 Internet Protocol version 6

MCUs microcontroller units

NAT Network Address Translation

NAT-PMP NAT Port Mapping Protocol

P2P Peer-to-Peer

PCP Port Control Protocol

PEX Peer Exchange

PM Policy Machine

RBAC Role-Based Access Control

SSI Self-Sovereign Identity

STUN Session Traversal Utilities for NAT

TURN Traversal Using Relays around NAT

UPnP Universal Plug and Play

VoIP Voice over Internet Protocol

WSN Wireless Sensor Network

XACML Extensible Access Control Markup Language

XML Extensible Markup Language

CONTENTS

1 INTRODUCTION

More and more the physical and digital worlds are merging into one. Sensors, actuators, and embedded devices are increasingly present in factories, smartphones, and supermarkets, changing the way we realize and interact with the environment around us. Recently, the advancement of various technologies, such as data analysis, cheaper sensors, Internet communication, and embedded systems enabled the emergence of a new revolution called the IoT. With these advancements, market interest and the use of the technology have been growing a lot in recent years, according to International Data Corporation, the number of IoT objects should reach 41.6 billion connected units generating a total of 79.4 zettabytes of data in 2025 (IDC, 2019).

With the exponential growth of data and devices, we need to ensure sufficient connectivity and resources for processing, collecting, transmitting, and exchanging data. Devices used in IoT are mostly constrained in memory, CPU, and power resources (BORMANN; ERSUE; KERANEN, 2014) and are generally composed of sensors and actuators. IoT platforms are frameworks that enable communication, integration, and decision coordination, allowing devices to transform from traditional objects to connected and intelligent agents.

Biology-based approaches such as Swarm Computing look at new classes of problems, in which task distribution, specialization, and decentralization of agents create more robust and flexible systems. Swarm Computing is a paradigm where computational devices start to collaborate allowing collective intelligence to emerge. In biology, swarm intelligence is used in reference to colony-level behavior for example, individual fire ants (*Solenopsis invicta*) struggle and drown after falling into a pool of water, however when the are in groups, fire ants link together to float on the water surface (MLOT; TOVEY; HU, 2011).

Inspired by organic systems, Swarm Computing enables the sharing of resources between IoT devices in a global network, in an autonomous and distributed way (LEE et al., 2014; COSTA et al., 2015b). Swarm-based systems are composed of thousands of smaller agents where the interactions among them create a collective behavior (COSTA et al.,

2015b). But these advantages come at the cost of complexity. In a non-scalable system, the addition of new agents can cause failures to appear after deployment (BJERKNES; WINFIELD, 2013), which compromise the entire functioning of the system. To guarantee the system's functioning, an optimized design with test and measurement techniques is necessary. Assuming the value generated from this network grows with the number of connections since having more agents means more features in the network, challenges arise in the scalability of these systems.

Concept of scalability can be defined as the work or cost of adding new resources to an existing network (BONDI, 2000), that is, it is generally said that a system is not scalable when the resources used for the growth of the system is excessive.

As intelligence emerges from the interaction and cooperation of the agents (LIU; PASSINO, 2000). The research proposal on the analysis of scalability techniques will be based on SwarmOS, a framework that emphasizes the communication of a large number of devices, making the scalability of IoT/Swarm architectures a key research point. SwarmOS provides open APIs, protocols, and tools for communication and operation within a network, facilitating the development of applications and ensuring interoperability between heterogeneous applications (COSTA et al., 2015b).

Utilizing the concept introduced by Lee and the TerraSwarm Research Center (2012) of a unified operating system (SwarmOS) supporting various "swarm applications," we can devise scalability testing methods. This information can then be employed to automatically optimize SwarmOS parameters, addressing scalability challenges. The optimization goals include enhancing message throughput, minimizing data transmission errors in geographically dispersed networks, and improving connectivity between diverse networks.

On the subject of scalability, there are researchers that developed in IoT, but the number is smaller in the context of Swarm. In IoT, as the study is often developed on top of a centralized system, they are focused on performance tests on test platforms (testbeds) and protocol tests to assess advantages in each use case. In the Swarm context, it is common for each agent to make decisions focused on communication and local sensing, so many researchers propose scalability as an inherent characteristic of this type of system, but this is not necessarily true as described by Bjerknes & Winfield (2013) who demonstrates that it is not safe to assume that scalability is an inherent attribute of swarm systems.

1.1 Motivation

In the context of a distributed IoT, such as the Swarm, the connection of millions, or even billions of devices is expected. In swarm systems that rely on cooperative behavior among nodes, scalability is essential to ensure that the network can sustain and benefit from the contributions of all its participants. Swarm also has an open architecture, where anyone can join and we can dynamically recruit resources, it will open up significant scalability problems, enabling new applications not yet invented or impossible right now to be created in the future.

1.2 Scalability Requirements

Because of the unique characteristics of IoT, with constrained devices having dynamic connections between them, scalability in IoT has different requirements:

- **Intermittent connectivity** – IoT devices can have intermittent connectivity because they move, are connected in fields with low Wi-Fi availability, etc. Thus, scalability must be able to be tested with many network characteristics in mind.
- **Dynamic registration of IoT entities** – IoT devices have become increasingly popular and are growing in number. We must guarantee that new devices being added to the network are accepted and ready to work without any problems.
- **Consideration for resource constraints** – IoT devices are resource-constrained (SEHGAL et al., 2012), for example, they can have a low CPU processing power, we must ensure that the connectivity overhead is minimal , and if tests are made using virtualization, they need to consider these characteristics.
- **Resilience and Robustness** – In some cases, availability is needed, resilience and robustness are essential to ensure that the network can continue to function even if some devices fail or become unavailable.
- **Locality** – Not depending on central or remote systems can mitigate significant latency and bandwidth issues, making it easier to scale the system effectively. It is possible to minimize the demand on the network by processing data locally and reducing the need for data transfer.

- **Ease of deployment** – Lowering the difficulty of deployment enables organizations to rapidly and cost-effectively deploy new devices and expand the network. If deployment is difficult or time-consuming, it can slow down the growth and expansion of the network, making it difficult to scale effectively.
- **Frequent communication between devices** – In the swarm case, communication between devices is a fundamental characteristic that emerges from the swarm intelligence, so dynamically varying situations resulting from mobility are expected from the network.

1.3 Challenges

This section discusses why the state-of-the-art does not cope with Swarm Computing requirements.

1. *Heterogeneity*: modern IoT platforms have a variety of devices and applications, the testbed has to assume heterogeneity is a characteristic of the system. Even if a project today has only one device model with one application running, because of the fast pacing of IoT, it's highly probable that with upgrades, the network will be composed of heterogeneous devices.
2. *Decentralization*: many state-of-the-art works assume only architectures where devices have a clear job of data collection, like in Wireless Sensor Networks Wireless Sensor Network (WSN)s when testing for scalability, it's assumed that the sensors are not a problem and the only tested part is usually the gateway receiving all the data. This is not true for other types of architectures, such as the Swarm.
3. *Limited Scope*: physical testbeds usually are focused on a single use case (smart city, WSNs, structural monitoring, low-level protocols), making it harder for research with different goals to be done on the same testbed.
4. *Test Scalability*: another way of testing done in the state-of-the-art is to set up physical testbeds, making it harder to scale to a sizable number of devices without being too expensive.

1.4 Contributions

In this dissertation, the SwarmOS is evaluated from a scalability point of view, breaking down the scalability factor and its challenges. The key contributions of the dissertation are summarized as follows:

- This dissertation helps **break down scalability and its challenges**. It maps the key enabling technologies for building scalable swarm networks. By exploring these technologies in the context of the SwarmOS, this dissertation provides a deeper understanding of how to achieve scalability in IoT systems. Furthermore, this research identifies the potential limitations and challenges associated with these technologies, as well as implement and/or propose solutions.
- **Evaluate the scalability of a swarm computing-based system**, assessing the scalability of centralized systems can be simpler, since there is a single point of control, meaning that all requests and data processing are handled by a central server; thus the scalability of a centralized system can be measured by the scalability of the central server. The scalability of a swarm computing-based system is interdependent of network and application scalability. Because of the high heterogeneity of the Swarm, there is no single benchmark that measures the scalability of the system, this work tries to **propose and specify test cases to measure scalability on a Swarm environment**.
- This work deployed **a concrete implementation of a testbed** for the SwarmOS, with high flexibility, being usable in various other use cases.
- **Improve SwarmOS from a scalability point of view**, investigate, propose, implement, and test changes to make the SwarmOS more scalable.

1.5 Structure of the dissertation

The structure of this dissertation is delineated as follows: Chapter 2 delves into the background and surveys related research works. Chapter 3 outlines the methodology employed for scalability tests across all components discussed in this study. In Chapter 4, we introduce and detail the Edge-computing IoT Testing framework, a key contribution of this dissertation. Chapter 5 provides a comprehensive overview of SwarmOS, the central framework subjected to scalability tests. Chapter 6 focuses on our evaluation of

SwarmOS's scalability, and, finally, Chapter 7 draws conclusions and charts out potential directions for future research.

2 BACKGROUND AND RELATED WORK

There is no single test capable of knowing if a system is scalable or not, scalability has many interpretations, depending on the situation or scenario, also there are many technologies and dimensions that can affect scalability.

In our pursuit of understanding scalability, this section unfolds with a multifaceted investigation into the global interconnection of devices, a paramount consideration in contemporary IoT frameworks. The intricacies of Network Address Translation (NAT) and its traversal techniques are scrutinized, alongside an insightful comparison of IPv6 and IPv4 performance, offering a holistic perspective on addressing the challenges of connecting devices across diverse networks.

Moving forward, our exploration delves into service discovery strategies, encompassing the innovative distributed registry system and a detailed examination of the BitTorrent model. These discussions pave the way for an exploration of cooperation incentives, where we navigate the realms of blockchain technology, microeconomics, and reputation systems as pivotal mechanisms driving collaborative dynamics in IoT ecosystems.

Furthermore, we turn our attention to the vital aspect of privacy, investigating Attribute-Based Access Control as a means of safeguarding sensitive information. The exploration culminates in an examination of the cutting-edge concept of Self-Sovereign Identity, underscoring the evolving landscape of privacy in the IoT paradigm.

By navigating through these diverse dimensions, this section aspires to provide a comprehensive backdrop for understanding the intricate interplay of factors influencing the scalability and global connectivity of devices in the IoT realm.

2.1 Scalability

The word scalability can have many meanings depending on the context, the Merriam-Webster dictionary definition is “capable of being easily expanded or upgraded on-demand”

(MERRIAM-WEBSTER, 2021) or if we use a computer science context, scalability means work or cost of adding new resources to an existing network (BONDI, 2000), these resources can be defined in various ways such as financial resources, human resources (time to set up, deployment in geographically distributed locations, etc.) and computational resources (throughput, CPU/memory usage, network resources, etc.). In this context, scalability refers to the capacity to handle a growing number of devices and users, to manage and transfer an increasing amount of data efficiently, and to support and enable new applications and use cases.

Scalability is a critical factor in the design and development of IoT systems, which often consist of a large number of interconnected devices, sensors, and actuators that collect, process, and transmit data. To ensure the efficient operation of these systems, the systems must be designed to scale with the growing number of devices. Gupta, Christie & Manjula (2017) presented research gaps. Protocol and network security, identity management, fault tolerance, access control, trust, governance, etc.

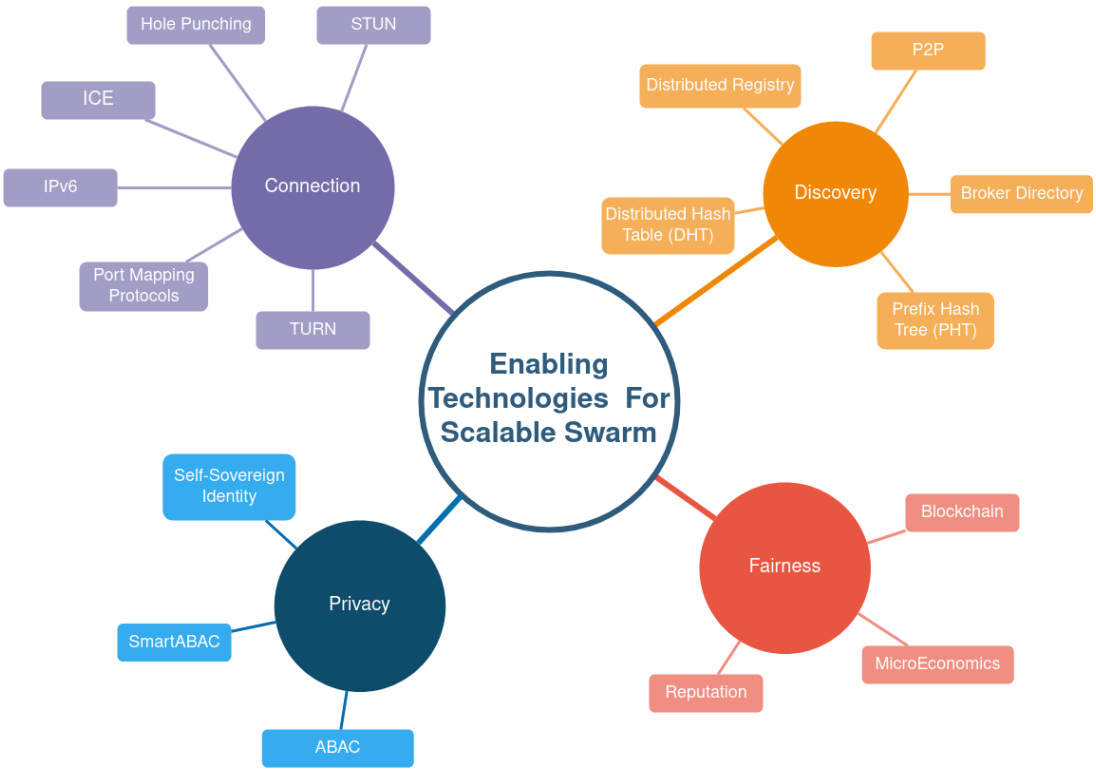
IoT scalability is categorized in, vertical and horizontal. The vertical scalability (SOTIRIADIS et al., 2016; CABRÉ; PRECUP; SANZ, 2017) refers to adjusting the computing resources of a single IoT device, while horizontal scalability (WU; LIANG; BERTINO, 2009; CHANG; SRIRAMA; BUYYA, 2019; SARKAR et al., 2014) is the addition or removal of IoT nodes to work on the same problem. In this work we will focus on horizontal scalability, considering four aspects: (1) connection, (2) discovery, (3) fairness, and (4) privacy. Figure 1 shows enabling techniques for a scalable swarm.

The connection aspect will evaluate the ability of Swarm agents to seamlessly link services across geographically distributed nodes without the need for centralized intermediaries and regardless of their limitations or to their physical network. In the Swarm, each node can act as both provider and consumer of services, enabling direct communication and data exchange between peers. This decentralized approach fosters efficient and resilient connections, promoting faster data dissemination, improved fault tolerance, and reduced latency.

The discovery aspect refers to the ability of each swarm agent to dynamically locate and interact with the available services at the moment. It enables nodes to query and discover other peers offering specific services or resources without relying on centralized servers. Service discovery facilitates seamless communication and collaboration within the decentralized network.

Fairness refers to the mechanisms and strategies employed to encourage nodes within

Figure 1: Feature Tree



the network to collaborate and share resources. As Peer-to-Peer (P2P) networks rely on decentralized interactions, cooperation incentives are crucial to foster mutual support and promote the sharing of services and data. These incentives can take various forms, such as reputation systems, where nodes with a history of cooperation are rewarded with increased privileges or better access to resources; or token-based economies, where nodes are incentivized to contribute resources by earning tokens or cryptocurrencies in return. Effective cooperation incentives play a vital role in maintaining network stability, scalability, and reliability, driving sustained collaboration among nodes.

Privacy answers the question if swarm agents can control their own identity and coordinate their interactions without having it stored and managed by a centralized third-party. It also empowers the owners to determine who has access to their devices. This decentralization means that the owner can not only restrict access to their devices but can also selectively grant permissions. For instance, while they can choose to keep their devices off-limits to most, they could allow specific individuals or entities access based on trust or other criteria. This granularity in control emphasizes the importance of privacy in ensuring both security and flexibility in the interactions within the swarm network.

2.2 Connecting Devices Globally

The original design of the Internet was based on a client-server architecture (GLOWNIAK, 1998), meaning that the client-side (personal computers, smartphones) requests resources, information, and data from servers (web servers, email servers, databases). This architecture was intended to allow for centralized control of the network and clear separation of responsibilities between client and server. This means that the client must start the communication and the server is only able to answer.

With the growth of P2P networks (e.g. torrents, Voice over Internet Protocol (VoIP), Swarm Computing), where all nodes can act as both clients and servers some complexity arises since the communication among nodes in P2P networks is dynamic and can occur between any two nodes, however, this P2P connections are not possible directly over the current Internet, because it is built over a NAT.

It is not straightforward to start a connection between two peers. Many network applications want a more P2P network, but to enable these connections between peers, we must understand how NATs works, and some of the most well-known NAT traversal techniques are required.

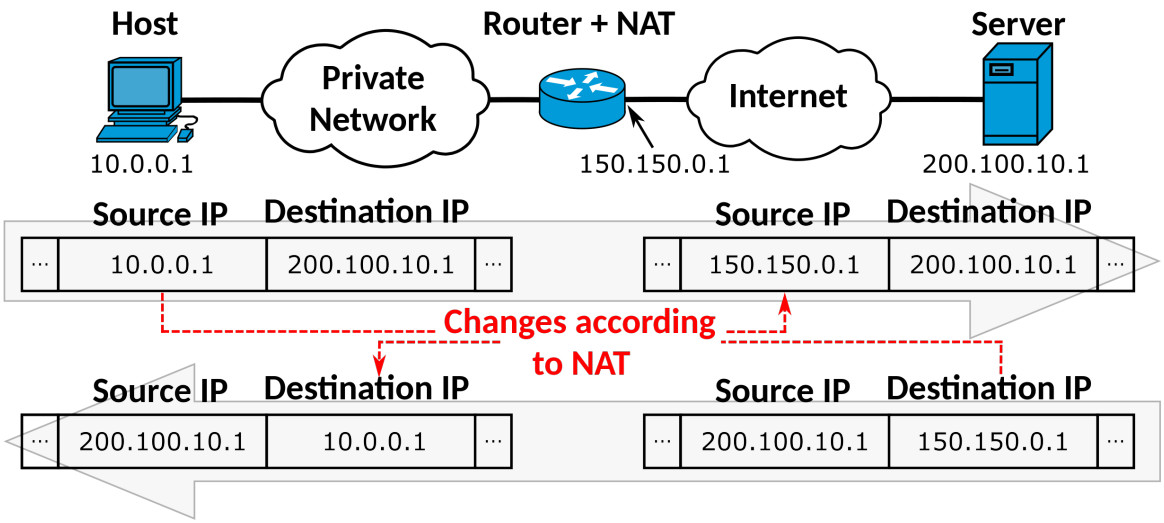
2.2.1 Network Address Translation

In order to gain insights into the constraints associated with sending messages over the Internet to any recipient, a thorough understanding of NAT is essential. NAT is a method of mapping an IP address space into another by modifying information in the IP header of the packets that are in transit across the router. This technique becomes popular because of Internet Protocol version 4 (IPv4) address exhaustion, where the number of IPv4 addresses cannot supply all existing devices.

Figure 2 depicts an example of a network address translation between the private network and the Internet. The client (with a private IP of 10.0.0.1) wants to send a packet to the server (200.100.10.1), the router with NAT capabilities changes the IP header of the packet to the public IP address (150.150.0.1), typically assigned by an Internet Service Provider. When the packets arrive at the server, the server answers, not even being aware of this modification. When the packet arrives at the router, the router changes the IP back to the private IP address and forwards the packet. This process is completely invisible to both the server and the client.

There are several types of NAT. Since there is no standardization between NATs

Figure 2: NAT IP address swapping. Source: Wikipedia contributors (WIKIMEDIA, 2020b)



(FORD; SRISURESH; KEGEL, 2005), each company or each model can be different from each one, but there are defined classes. In Figure 3 we can see 4 categories of NAT that this dissertation will dive deeper to understand why the connection can be difficult.

We are using the following terminology: Every address will be defined by a pair of IP addresses and Port, (Addr:Port).

iAddr:iPort : Internal address and internal port, means the address and port of the client in the private network.

eAddr:ePort : External address and external port, means the address and port of the client after NAT. Usually the address that the Internet service provider assigned.

sAddr:sAddr : Server address and server port, means the IP and port of the server, who we are trying to communicate with.

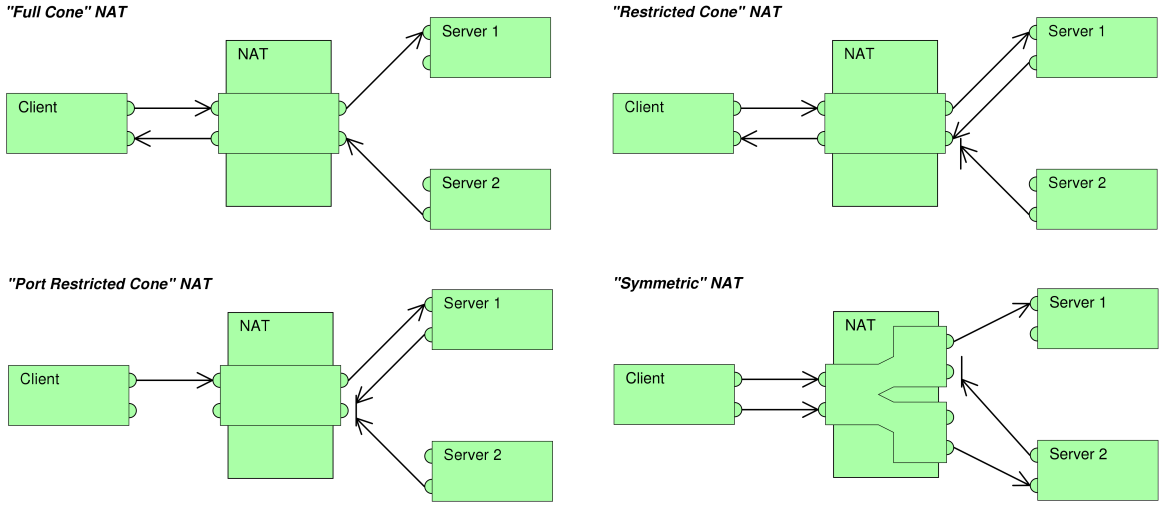
Any : "Any" means that the IP or port value does not matter.

Full-cone NAT

Once an internal address (iAddr:iPort) is mapped to an external address (eAddr:ePort), any packets from iAddr:iPort are sent through eAddr:ePort. All subsequent incoming packets to iAddr:iPort by are redirected to eAddr:ePort. That means, full-cone NAT allow messages coming from *any*:*any*

Address-restricted-cone NAT

Figure 3: Examples of 4 categories for NATs, Full Cone, Restricted Cone, Port Restricted Cone and Symmetric NAT. Source: Wikipedia Contributor (WIKIMEDIA, 2020a; WIKIMEDIA, 2020d; WIKIMEDIA, 2020c; WIKIMEDIA, 2020e)



Once an internal address (iAddr:iPort) is mapped to an external address (eAddr:ePort), any packets from iAddr:iPort are sent through eAddr:ePort. An external server (sAddr:any) can send packets to iAddr:iPort by sending packets to eAddr:ePort only if iAddr:iPort has previously sent a packet to sAddr:any. That means incoming traffic can arrive from any port, as long as it originates from the expected external host.

Port-restricted-cone NAT

Once an internal address (iAddr:iPort) is mapped to an external address (eAddr:ePort), any packets from iAddr:iPort are sent through eAddr:ePort. An external host (sAddr:sPort) can send packets to iAddr:iPort by sending packets to eAddr:ePort only if Addr:iPort has previously sent a packet to sAddr:sPort. That means that incoming traffic is allowed only from the expected external host and on the specific port used for the initial communication.

Symmetric NAT

Each request from the same internal IP address and port (iAddr:iPort) to a specific destination IP address and port is mapped to a unique external source IP address and port (eAddr:ePort); if the same internal host sends a packet even with the same source address and port but to a different destination, a different mapping is used. Unlike other NAT types, the mapping changes for each destination, making it difficult for external hosts to predict the port used by the internal device. This behavior restricts inbound traffic from reaching the internal device unless the internal device specifically initiates

communication with the external host.

2.2.2 NAT Traversal Techniques

Understanding NATs, we can see that it is not possible for a device to just send another device packets. Even if they know the Addr:Port of their peer, the packet will likely be dropped by the NAT. So to establish and maintain a connection that works across these NATs, traversal techniques are required. Table 1 shows a simple survey of different types of NAT traversal techniques.

There are three main solutions categories for this issue:

- **Hole Punching**: Send packets to open holes in the port mapping of the NATs. This strategy works even when there are multiple NATs between the peer and the Internet, unless there is a symmetric NAT in the way.
- **Port Mapping**: A peer asks for its NAT to create a port mapping from its private network to the Internet. However, this solution only works in cases where there is only 1 NAT between the peer and the Internet. In some cases, an Internet Service Provider can take a whole region and use a NAT to share the same IP with every customer. In this case, the port mapping will not expose the true public IP, but a private regional IP.
- **Relaying**: Utilizing a central server accessible to both peers, acting as an intermediary between Peer1 and Peer2. In this approach, data packets traverse the network through the following path $Peer1 \leftrightarrow Server \leftrightarrow Peer2$. While this method proves effective in numerous scenarios, it comes with the drawback of centralization and inefficiency. The reliance on a high-bandwidth server for continuous packet relay introduces wasteful resource consumption.

Hole Punching

Hole punching enables two clients to set up a direct connection with the help of a well-known rendezvous server, even if the clients are both behind NATs.

Usually, a NAT is configured as an outbound NAT, which means that it will send packets from inside the private network to the Internet but will drop any external requests, except packets that are answers from previous requests.

Table 1: Simple survey of different types of NAT traversal techniques

Name	Type	Pros	Cons
NAT Hole Punching	Hole punching	Works with Transmission Control Protocol and User Datagram Protocol (Transmission Control Protocol 64%, User Datagram Protocol 82%) (FORD; SRISURESH; KEGEL, 2005)	Hairpin translation, a feature that permits the access of a service via the public IP address from inside the local network, is less common (FORD; SRISURESH; KEGEL, 2005)
Traversal Using Relays around NAT (TURN) (REDDY.K et al., 2020)	Relay	Works in any case	High bandwidth needed on the server Add latency on the communication Both services needed to connect on the same server
Session Traversal Utilities for NAT (STUN) (ROSENBERG et al., 2003)	Hole punching	Works in 3/4 of types of NATs Direct connection On swarm maybe we can use registry as Session Traversal Utilities for NAT (STUN) server	Don't work with symmetric NATs
Interactive Connectivity Establishment (ICE) (KERäNEN; HOLMBERG; ROSENBERG, 2018)	Hole punching, Relay	Tries many connections and choose the best one based on heuristics	Worst case scenario use a Traversal Using Relays around NAT (TURN) server
UPnP Internet Gateway Device Protocol (IGDP) (BOUCADAIR; PENNO; WING, 2013)	Port mapping	Direct Connection	Only works if you NAT is connect directly to the public internet NAT has to implement this protocol Risk of security opening a port to the internet
NAT Port Mapping Protocol (NAT-PMP) (CHESHIRE; KROCHMAL, 2013)	Port mapping	Direct Connection	Only works if you NAT is connect directly to the public internet NAT has to implement this protocol Risk of security opening a port to the internet
Port Control Protocol (PCP) (BOUCADAIR; PENNO; WING, 2014)	Port mapping	Direct Connection	Only works if you NAT is connect directly to the public internet NAT has to implement this protocol Risk of security opening a port to the internet

The hole punching is a way of exploring this exception, with the use of a rendezvous server. Both clients communicate with the rendezvous server that will answer with the external address of the target peer. After getting to know the public IP of each other, both send messages to each other. When peer1 with Addr1:Port1 sends a message to peer2 on Addr2:Port2, his NAT (NAT1) opens an exception, accepting outbound packets from Addr2:Port2 because is a response to a previous request, the same can be said the other way around. If both parties transmit messages within a given time frame, typically a few seconds, the ability to initiate communication and exchange data between them is established. However, since NAT implementations lack standardization, the specific duration of this time frame is not defined. Ford, Srisuresh & Kegel (2005) goes into more detail on how it works and tests in which case it works.

Port Mapping Protocols

Port mapping or also known as port forwarding is when an application sends a message to the NAT that redirects a communication request to create an Addr:Port pair that traverses the network gateway. For port mapping protocols, it is required that your NAT is connected directly to the Internet.

Examples of these protocols are UPnP Internet Gateway Device Protocol (IGDP) (BOUCADAIR; PENNO; WING, 2013), NAT Port Mapping Protocol (NAT-PMP) (CHESHIRE; KROCHMAL, 2013), Port Control Protocol (PCP) (BOUCADAIR; PENNO; WING, 2014)

Universal Plug and Play (UPnP), NAT-PMP, and PCP are all networking protocols designed to facilitate port mapping for NAT devices.

UPnP and NAT-PMP are distinct implementations of a common concept, wherein an endpoint requests the router or NAT device to open a port for external data reception. Initially developed by Apple, NAT-PMP was later adopted as an IETF standard (CHESHIRE; KROCHMAL, 2013), while UPnP is an ISO/IEC standard (ISO/IEC..., 2017). In its RFC (CHESHIRE; KROCHMAL, 2013), NAT-PMP's section 9 outlines the reasons for not using UPnP, citing advantages such as simplicity, focused scope, efficiency, garbage collection, and other attributes. Later PCP came as a successor of NAT-PMP, PCP added support for Internet Protocol version 6 (IPv6) and additional NAT scenarios (CHESHIRE et al., 2013).

The functional requirements of these port mapping protocols indicate that it only works when the router is directly linked to the Internet, and not configured behind another NAT device. Furthermore, successful implementation of this protocol is necessary on the NAT device supporting the protocol being implemented, because the standardization of

Table 2: NATs' type and relative traversal capability of STUN. Source: Wang *et. al.* (WANG; LU; GU, 2006)

Class	NAT 1 Type	NAT 2 Type	Traversal capability
1	Full Cone	Any	YES
2	Address or Port restricted	Address or Port restricted	YES
3	Symmetric	Address or Port restricted	NO
4	Symmetric	Symmetric	NO

PCP, NAT-PMP and UPnP are recent (PCP, NAT-PMP in 2013 and UPnP in 2008) it is possible to find routers that still don't implement these standarlizations. It is crucial to consider the inherent security risk associated with directly exposing ports to the Internet when utilizing this protocol, as it can leave devices and services vulnerable to potential external threats.

Session Traversal Utilities for NAT

Session Traversal Utilities for NAT (STUN) (ROSENBERG et al., 2003) is a lightweight protocol that allows applications to discover the presence and types of NATs between them and the public Internet. It also provides the ability for applications to determine the public IP addresses allocated to them by the NAT.

Wang et. al. (WANG; LU; GU, 2006) shows that STUN protocol cannot effectively support Symmetric NAT.

Traversal Using Relays around NAT (TURN)

TURN (REDDY.K et al., 2020) allows the host to control the operation of the relay and to exchange packets with its peers using the relay. The difference TURN provides from other relay control protocols is that it allows a client to communicate with multiple peers using a single relay address. TURN is used in many other protocols since using relays the relay server needs a high bandwidth to keep relaying packets, usually, TURN is used as a fallback in case all other protocols fail.

Interactive Connectivity Establishment (ICE)

ICE (KERäNEN; HOLMBERG; ROSENBERG, 2018) is a NAT traversal technique, it uses heuristics to choose which protocol to use, it mixes STUN and TURN. ICE works by exchanging a multiplicity of IP addresses and ports, which are then tested for connectivity by P2P connectivity checks.

2.2.3 IPv6

IPv6 is the successor to IPv4, which is a widely used protocol for communicating over the Internet (IETF, 1998). IPv6 was designed to address the limitations of IPv4, such as the limited number of available addresses. This protocol introduces several new features and improvements, such as larger address space, improved packet handling, and support for new mobility features.

IPv4 provides an addressing capability of 2^{32} or approximately 4.3 billion addresses. The anticipated shortage has prompted the development of new technologies, such as NAT. IPv6 provides many advantages for connecting devices globally compared to IPv4.

IPv6 increases the address space from 32 bits to 128 bits, allowing for an enormous number of unique addresses. This larger address space will be crucial in the future as the number of devices connected to the Internet continues to grow exponentially.

Secondly, IPv6 supports auto-configuration, which enables devices to configure themselves with an IP address without requiring manual configuration or a DHCP server. This is particularly useful for IoT devices that are frequently added or removed from a network. Additionally, IPv6 includes new mobility features, such as seamless handovers between different networks and the ability to maintain connections while moving between networks.

2.2.3.1 IPv6 vs IPv4 Performance

One pressing question to consider is whether switching to IPv6 guarantees the same level of performance. Will the transition maintain the status quo, or will there be a noticeable enhancement or decline in the network's functioning?

According to Zhou et al. (2008), there are legitimate concerns surrounding the performance and scalability of IPv6 in comparison to its predecessor, IPv4. While native IPv6 paths exhibited delays comparable to IPv4, the introduction of IPv6 tunnels significantly prolonged these delays. Particularly worrisome were the delays tied to IPv6-in-IPv4 tunnels.

Several factors contribute to the scalability issues observed with IPv6:

1. **Infrastructure Limitations**: A network with a limited number of routers tailored for IPv6 hardware could lead to diminished performance.

2. **Routing Path Challenges**: There's a tendency for IPv6 to utilize suboptimal routes, a problem accentuated when tunnels are in use.

3. **Network Management Shortcomings**: IPv6 networks seem to be a step behind in terms of advanced management and monitoring when juxtaposed with IPv4 networks. This gap arises partly because many ISPs are more reluctant to heavily invest in IPv6 management, and also because of the extensive operational history associated with IPv4.

4. **Priority Issues**: IPv6 is sometimes relegated to a lower priority status in routers, often viewed as a more experimental technology, thereby potentially compromising its efficiency.

Additionally, when it comes to packet loss metrics, IPv4 still has an edge over IPv6. It's widely believed that both IPv4 and IPv6 will operate in tandem for an extended period, with IPv6 tunnels facilitating the transition. But, the data suggests that an over-reliance on these tunnels adversely affects network performance. To achieve the best from IPv6, a blend of native IPv6 and hardware-centric routers is ideal. These findings emphasize the need for a well-thought-out infrastructure approach, considering IPv6's inherent constraints, to assure a seamless and proficient network experience.

2.3 Service Discovery Strategies

Bröring, Datta & Bonnet (2016) categorizes service discovery strategies based on where the search is done. We can categorize them as searching physically around the device, locally on the network, or globally outside the local network.

For discoveries focused outside the local network, Ccori et al. (2016) used an agent-based simulation to evaluate centralized, decentralized, and hierarchical networks based on some criteria. In centralized networks, we have two agent types, the central node, and the services nodes. Decentralized network every node is a service where they have an average number of neighbors known by each other. Hierarchical networks have supernodes and service nodes, every service node is connected to a supernode that is connected with other supernodes.

Based on (CCORI et al., 2016), the hierarchical topology gives us a good balance between the criteria evaluated, since the data transmitted is not too big, only for the discovery of new services, the hierarchy can be used to bootstrap communications faster,

similar to trackers are used on the BitTorrent protocol (WU et al., 2010). A criteria not discussed on (CCORI et al., 2016) is that for discovery central and hierarchical solutions have the advantages of being simpler to implement and having advanced querying and ranking capabilities, for example, semantic querying.

One common mechanism for peer discovery is to use a tracker. When a peer joins a torrent, it typically registers with one or more trackers. Any peer can contact a tracker at any time to obtain a random subset (IP-port pairs) of other peers in the torrent. Today, there are at least a dozen tracker implementations. Many BitTorrent clients also support "distributed trackers" using Distributed Hash Table (DHT)s and Peer Exchange (PEX).

2.3.1 Distributed Registry

A distributed registry refers to a database or data storage system that is distributed across multiple nodes in a network. In this type of system, data is stored on multiple computers or servers instead of being stored on a single central server. Each node in the network has a copy of the part registry and can be used to access and update the data stored in it. This type of registry is often used in decentralized networks and is designed to be highly available, fault-tolerant, and scalable. The main advantage of a distributed registry is that it eliminates the need for a single point of failure and allows for the distribution of processing and storage load across the network. One notable drawback is the potential for increased complexity in managing data across multiple nodes. Synchronization issues between distributed nodes may arise, leading to consistency challenges.

2.3.1.1 Distributed Hash Table

One of the ways of providing a service for discovery and lookup is to use a DHT. A DHT is a type of distributed system that provides a service similar to a hash table data structure, which means key-value pairs are stored and can be retrieved. Each DHT node has a unique identifier and is responsible to store and keep part of the key-value of the system. When a new lookup request is made on the system, a routing algorithm resolves which node is responsible for storing that key and sends a request to that node. This allows efficient data storage since a central server is not needed to manage the entire database, another advantage is that each node only needs to host part of the key-value pairs, making it lightweight. Also, another advantage of a DHT is that nodes can be added or removed with minimum work around re-distributing keys.

With the growth of decentralized networks, finding a specific resource in a scalable manner is a real challenge. There are many works using DHTs to leverage their key-value pairs as a way to locate a specific service (SARMADY, 2010).

Although the DHT field has not reached its full potential and lots of new research improving DHTs was published in the last few years, we have several reliable DHT implementations (STOICA et al., 2001; RHEA et al., 2003; ROWSTRON; DRUSCHEL, 2001) are readily available. Additionally, there are multiple DHT applications that are in daily use (FREEDMAN; FREUDENTHAL; MAZIERES, 2004; MISLOVE, 2003; RAMASUBRAMANIAN; SIRER, 2004), and even more ideas have been proposed and/or have prototypes.

2.3.2 The BitTorrent System

In the context of BitTorrent, multiple users (peers) share files with each other. Each peer in the swarm acts as both a client and a server, downloading and uploading pieces of the shared files to other peers.

Understanding how scalability works on one of the most used P2P protocols proved useful to understand and find technologies to improve the SwarmOS.

Mendes et al. (2017) made a great experimental model to assess the throughput of P2P systems, he uses a BitTorrent client to test the throughput of the systems and compare it with the theoretical findings of other researchers. Since the test was only done with the BitTorrent client, only one service (content distribution) was tested, so the work is not suited for all systems, but it shed some light on incentive strategies to increase the scalability of the system. Silva et al. (2019) presents new results to quantify the throughput of P2P swarming systems based on different neighbor and block selection policies.

Piatek et al. (2007) studies whether incentives help the throughput of BitTorrent and discover that tit-for-tat strategy works but the bulk of the performance effect in practice is altruistic contribution on the part of a small minority of high-capacity peers. And selfishness can hurt Swarm performance.

Stutzbach, Zappala & Rejaie (2005) shows using a modeling simulation that swarm systems scale positively with their size, where the system capacity increases with the number of peers participating.

Besides the tit-for-tat strategy, another way to give incentives is to give a “credit” to all people who give more to the network, if a user doesn’t provide a reasonable ratio, he

can be expelled from the network. Hales et al. (2009) studied this case and showed that a small minority of peers can obtain a large amount of credit in the system. So adding this incentive to the system can, counter-intuitively, reduce the throughput of the whole network due to a shortage of credit.

2.4 Cooperation Incentives

Intrinsic motivation alone may be insufficient to consistently drive resource sharing among participants. To incentivize resource sharing among participants in a P2P network, extrinsic motivators are necessary, Bogliolo et al. (2012) divide the forms of rewards into three. *Reputation*, which improves the ranking of an individual in a group; *reciprocity*, making an individual agent first cooperate in the likelihood of future mutual cooperation; and *monetization* which is to assign value to services. These incentives can be viewed as a reward to help align individuals' utility to network utility and penalize nodes that engage in non-cooperative behavior.

To keep users motivated to share services and resources (bandwidth, computational power, storage space, etc.) and to avoid malicious nodes inhibiting the functioning of the whole system it is necessary to implement incentive mechanisms. Virtual currency is often adopted in online communities to encourage participation, but by itself doesn't guarantee good quality of service (risk of dishonesty), which could keep people from taking pro-network decisions.

To reduce this risk, reputation mechanisms are commonly used in online communities. They can be used both to boost participation and keep the quality of service.

A reputation system can play a crucial role in enhancing scalability within a network by efficiently identifying and expelling bad agents. By assigning reputation scores to agents based on their behavior and interactions, the system can distinguish between trustworthy and untrustworthy entities. As a result, bad actors with consistently poor performance or malicious intent can be identified and expelled from the network, thereby preventing them from causing disruptions and reducing the overall network's efficiency. This proactive approach in managing agent reputations fosters a healthier and more reliable environment for interactions, allowing the network to scale more effectively and handle a larger number of participants with greater resilience.

2.4.1 Blockchain

To make the economic and reputation aspects more challenging, the distributed nature of these networks forces the adoption of a transparent and distributed registry to store this transaction information. Blockchain is a decentralized and transparent ledger that can securely store and manage transactions and data. This makes it an ideal tool for incentivizing cooperation in decentralized networks, where trust and security are crucial.

In recent years, the research of IoT applications has become increasingly complex. To address the hardware constraints of IoT devices and centralized models, the blockchain and P2P approaches can provide an improvement in scalability with the development of decentralized and data-intensive applications running on many devices.

Another way blockchain can be used as a cooperation incentive is through the creation of token-based incentives. Tokens can be used to represent various forms of value, such as monetary, reputation, or influence. In decentralized networks, tokens can be used to incentivize cooperative behavior by rewarding nodes for their contributions to the network.

But one question is how the blockchain performs based on scalability. For example, Bitcoin's blockchain is known for requiring a long time to store and preserve the consistency of the blockchain database, this blockchain has a low limit of only 7 transactions per second (CROMAN et al., 2016). Conoscenti, Vetro & Martin (2016) did a literature review and found some works that consider this case.

Wörner & Bomhard (2014) uses the blockchain to purchase sensor data via bitcoins. In this work, the author says there are scalability issues due to the exploding nature of the number of transactions. Zyskind, Nathan & Pentland (2015) comes to a similar conclusion. Another issue pointed out by Barber et al. (2012) is that every node should verify each block/transaction on the blockchain, making it inviable because of the constrained resources that IoT devices usually have.

2.4.2 Micro Economics

Incentives can also be provided in the form of monetary rewards. For example, in a swarm network nodes that provide high-quality service are rewarded with monetary compensation. This incentivizes nodes to provide high-quality services, ensuring that the network operates efficiently and effectively. Also, by providing a financial incentive for individuals or organizations to invest their time and resources into the development of new products or services, monetization can help drive innovation and spur technological

advancements.

By providing tangible benefits for cooperative behavior, monetization systems create an incentive for individuals to actively participate and contribute to the community. By generating revenue through usage, these systems provide a source of funding for continued development, maintenance, and growth. This, in turn, can attract new members and encourage further investment, leading to a virtuous cycle of growth and cooperation.

Biase et al. (2018) proposed a swarm microeconomy model for digital resource sharing that works for large-scale networks while fulfilling constraints created by the presence of low-resource devices. A blockchain-based transaction was developed for transparent and immutable currency audit thereby ensuring trading among untrusted users.

2.4.3 Reputation

Monetization by itself may not be enough to have a scalable network. In the work (BJERKNES; WINFIELD, 2013) on swarm robotics, the addition of fault agents can lower the overall system reliability. To earn money/tokens/coins the agent can have a low quality of service to improve their own profit.

Reputation systems play a crucial role in incentivizing cooperation in online communities by creating a mechanism for individuals to publicly display and build their trustworthy behavior, thus leading to a more harmonious and productive environment.

In networks, reputation systems serve as a crucial tool to establish trust and incentivize cooperation among participants. Reputation systems mitigate the likelihood of fraud by providing a public record of an individual's past behavior, allowing others to make informed decisions about interacting with them.

Reputation systems also provide incentives for individuals to engage in cooperative behavior by allowing them to build a positive reputation and increase their visibility within the community. This can lead to increased opportunities for transactions, as well as improved relationships with other members. On the other hand, individuals with a negative reputation face consequences such as reduced visibility and a decrease in opportunities for transactions.

Combining economical and reputation-based incentives we can benefit from complementary strengths to overcome their weaknesses. Together, these incentives can create a positive feedback loop where individuals are motivated to engage in cooperative and profitable behavior, leading to increased network activity, and improved overall trust in the

platform. By combining economic and reputation-based incentives, online marketplaces can create a more efficient and trustworthy environment that benefits all participants.

2.5 Privacy

When building a network, a common approach is to create a centralized service to govern interactions, manage identities, and query the network. However, as the network grows, this centralized approach can become a burden. The centralized service becomes a single point of failure, and with its expansion, it can become unable to handle the increased traffic and demand. This can lead to slow response times, outages, and frustrated users.

Because of that, a question that arises is, can swarm agents control their own identity and coordinate their interactions without having it stored and managed by a centralized third-party? One advantage of a decentralized approach is that it can be more scalable and reliable. As the network grows, new nodes can be added to the network without overwhelming the existing infrastructure. Additionally, there is no single point of failure, so the network can continue to operate even if some nodes fail or are taken offline.

To solve access control security, Attribute-Based Access Control (ABAC) offers a promising solution to address these concerns by providing a flexible, scalable, decentralized and fine-grained access control mechanism.

And to solve the identity management problem, a solution is to use Self-Sovereign Identity (SSI). SSI is a digital identity model that aims to give individuals full control over their personal data and identity information, empowering individuals with control over their own digital identities in the context of IoT ecosystems. SSI enables individuals to securely manage and share their personal information with IoT devices and services without relying on centralized authorities.

2.5.1 Attribute-Based Access Control

With the anticipated substantial increase in the number of consumer electronics devices per person (STANKOVIC, 2014), the protection of consumer electronics devices requires the implementation of access control mechanisms. Ensuring that only authorized entities can access specific devices becomes crucial. This becomes particularly challenging in scenarios like swarm, where users possess hundreds or thousands of devices at home, making the management of authorizations between these devices a usability challenge (LEE; KIM, 2017).

Access Control is vital in the Swarm, ensuring confidentiality, ownership, and protection against cyber-attacks by permitting only authorized communication.

When considering the implementation of access control in the context of IoT, traditional approaches like Access Control Lists (ACL) (e.g., (WANG et al., 2015; GAŁKA; MASIOR; SALASA, 2014)) and Role-Based Access Control (RBAC) encounter challenges related to scalability and manageability (HU et al., 2013).

Applying ABAC in Swarm systems provides a powerful and versatile approach to address the security and access control challenges prevalent in these complex environments. ABAC offers granular control over access decisions by considering various attributes such as user roles, device characteristics, location, and time. In Swarm systems, ABAC can play a crucial role in device authentication and authorization, ensuring that only trusted devices are granted access to the network and its resources. ABAC also enables fine-grained control over data and resource access, allowing administrators to define access policies based on attributes like data sensitivity, user context, and device capabilities. This facilitates efficient sharing and utilization of Swarm data while ensuring privacy and confidentiality. Real-time adaptation of access control policies in response to dynamic changes in the Swarm environment becomes possible with ABAC, providing scalability and flexibility.

Additionally, ABAC in Swarm systems enables centralized policy management and enforcement, simplifying administration and reducing the complexity associated with managing access control in large-scale deployments. The integration of ABAC in Swarm systems promotes robust security and helps mitigate the risks associated with unauthorized access, data breaches, and malicious activities, ultimately enhancing the overall trust and reliability of Swarm ecosystems.

In Fedrecheski et al. (2018), the author proposes an ABAC policy system that enables effortless policy creation and distribution across devices, eliminating the need for a centralized authorization server.

2.5.2 Self-Sovereign Identity

SSI is a digital identity model that aims to give individuals full control over their personal data and identity information (TOBIN; REED, 2016). In traditional identity systems, such as those used for online services or government-issued IDs, a third party is usually responsible for verifying and storing the identity data. However, in an SSI system, the

individual is the sole owner and controller of their identity data, which is stored on a decentralized network of nodes.

This decentralization allows for increased privacy, as personal data is not stored in a central location vulnerable to hacking or data breaches. SSI also allows for greater portability and interoperability of identity data, as individuals can choose which pieces of their identity to share with different organizations, without needing to rely on third-party verification.

To achieve SSI various technologies such as blockchain, public key cryptography, and Decentralized Identifier (DID) standards are used to create a secure, decentralized network that enables individuals to control their identity information. The goal of SSI is to create a new paradigm of identity that empowers individuals with greater control, privacy, and security in their online interactions.

3 METHODOLOGY

This research is designed as a study case of the SwarmOS scalability. It suggests and evaluates scalability strategies for edge based, decentralized, open IoT networks of devices. The research uses a mixed research method, relying mostly on quantitative approaches for the scalability analyses, designing new tools to support data gathering.

Data will be collected through simulations and experiments using IoT testbeds. These testbeds will replicate real-world Swarm scenarios to capture performance metrics and scalability characteristics. Additionally, qualitative analyses become necessary when simulations and experiments cannot achieve the required scale.

The study explore various iterations of the SwarmOS case, suggesting and implementing improvements. Specifically, SwarmOS 0.2.3 undergoes a scalability evaluation, leading to proposed enhancements that are subsequently implemented and rigorously tested

The SwarmOS scalability tests were conducted using a tailored-made framework, the **Edge-computing IoT Testing framework**, since the state of the art testbeds have limitations. For example, in many works, these platforms were purpose-built for particular use cases, often specialized to a singular protocol, network architecture, and/or data pipeline. Consequently, adapting these tools to work in different use cases can be laborious. Given the requirements of SwarmOS, which necessitates a platform capable of accommodating a wide array of protocols, architectures, and devices, the development of a novel framework with a strong emphasis on flexibility became imperative. The new framework is proposed and is presented in subsection 4.

The scalability analysis is conducted considering four aspects: (1) connection, (2) discovery, (3) fairness, and (4) privacy.

3.1 Connection

The connection tests were designed to analyze scalability regarding connections within the Swarm network. Specifically will be tested: connectivity through varied network environments, and concurrent connections capability.

3.1.1 Testing Device Interactions in Varied Network Environments

This test has the goal to determine whether the swarm broker can effectively manage connections across different networks.

Testbed Setup

The setup comprises a physical testbed, where physical devices are distributed across multiple types of networks, each with distinct configurations.

Performance Metrics and Data Analysis

The study was designed to measure the efficacy of communication between devices, which were pre-programmed with each other's service descriptions, across a wide range of network environments, from direct connections to public internet and home networks situated behind NATs.

Experimental Design

The test consists of 2 Labrador 64bit boards [1] from Caninos Loucos that were located in different networks. These were real commercial networks, physically separated and different ISPs. The first test was behind a working commercial NAT, while the subsequent test occurred on an internet infrastructure prepared for IPv6 connectivity. Each board has a broker and a dummy temperature service, while one of the boards also runs a blockchain agent. The test begins with both boards being aware of each other's service description, and they subsequently sought to use each other's service.

The test aims to determine if both devices can initiate a connection and successfully request temperature data, while also assessing the communication performance for any significant inefficiencies stemming from bugs or errors. However, a direct comparison of post-communication latency or delay was not conducted, as the modification of protocols like UPnP was expected to have an insignificant impact on measures such as delay or

[1] https://caninosloucos.org/en/labrador-64-en/

packet loss.

Scalability Evaluation

The evaluation of scalability was performed by measuring the error rate, assessing the frequency of errors and mistakes encountered during the setup and development phases.

Ethical and Security Considerations

The use of certain network protocols introduces security issues, especially if these protocols lack robust encryption, authentication, or authorization mechanisms. Vulnerabilities in poorly implemented or outdated protocols could potentially lead to unauthorized access, data breaches, and other security threats.

For example, the use of the UPnP protocol raises security considerations due to its inherent openness and simplicity, which exposes devices to potential vulnerabilities and unauthorized access. Security measures, such as proper authentication, encryption, and access control mechanisms, need to be carefully implemented to mitigate these risks and ensure a robust and secure IoT environment.

The researchers owned all the devices, data, and network involved in the experimentation, ensuring full control and compliance with ethical standards. Moreover, the testing phase was conducted without exposing any risks to individuals or external entities, prioritizing the safety and privacy of all stakeholders involved.

Limitations

P2P connections, harnessing a diverse array of network protocols, are instrumental for facilitating effective communication in the dynamic and diverse P2P environment. Despite the test being conducted with a restricted number of devices across a few networks, it yielded valuable insights. The focus on 1-to-1 connections within the network composition allowed for a targeted examination, offering significant insights into the performance and adaptability of protocols in specific scenarios. This deliberate approach ensures that the test remains valid, providing nuanced understanding and paving the way for potential adaptations tailored to specific network configurations.

3.1.2 Concurrent Connection Capacity Test

This test has the goal to determine the maximum number of simultaneous connections a swarm broker can handle without degrading performance or experiencing failures.

Testbed Setup

The configuration involves the **Edge-computing IoT Testing framework**, further described in Section 4, ensuring all devices are interconnected within a unified network. This network features a single consumer and several providers. The objective of the test is to ascertain whether one broker can simultaneously communicate with several others without encountering any failures.

Performance Metrics and Data Analysis

The test will measure several critical metrics, including the maximum number of simultaneous connections the broker can manage, the consumption of resources - CPU, memory, and bandwidth - at different connection thresholds, the response times observed at these levels, and any recorded error rates or instances of connection drops. After the test, it's essential to pinpoint the count of connections when the broker starts showing strain, marking it as its maximum capacity. The collected data will be analyzed to identify patterns in resource usage as connections escalate. Additionally, an assessment will be conducted to determine if the broker faces particular resource constraints, such as being CPU-bound or memory-bound.

Experimental Design

During the initial setup, the process begins by taking a baseline measurement when the swarm broker has no client connections. This stage also involves the careful monitoring and documentation of the broker's initial resource consumption, including metrics like CPU, memory, and bandwidth usage.

As we proceed to the incremental connection phase, connections to the swarm broker are initiated in a phased manner. This might involve batches of connections, which is contingent on the anticipated capacity. Following the establishment of each of these connection batches, there's a need to consistently monitor and log both the resource consumption and the broker's response times. Concurrently, it's essential to remain vigilant for any occurrences of connection interruptions, potential timeouts, or any errors that the client nodes might experience.

The subsequent phase is the determination of the stress point. Here, the objective is to continuously ramp up the number of concurrent connections. The progression halts when the swarm broker demonstrates any of the following: a pronounced decline in its performance, such as heightened latency or diminished throughput; a tendency to decline fresh connection attempts; or a marked rise in error occurrences or connection disruptions.

Scalability Evaluation

The expected result is to have a distinct understanding of the threshold for concurrent connections, beyond which the performance of the swarm broker begins to diminish. Gaining this insight is crucial for comprehending the scalability potential of the broker. Moreover, this knowledge can steer decision-making processes concerning load balancing or the clustering of brokers, especially in expansive setups.

Ethical and Security Considerations

All data, devices, and networks employed in this experiment were simulated, avoiding the need for ethical and security considerations.

Limitations

Although conducted entirely in a virtualized environment, this test holds significance due to the fact that the same code can be utilized between virtualized and physical devices. This consistency is a key strength, bolstering the test's reliability in uncovering bugs, limitations, or unexpected behaviors across diverse scenarios. The uniformity in code implementation ensures that the insights gained from the virtualized environment are highly applicable to real-world situations, making the test a robust and valuable component of the overall evaluation process.

3.2 Discovery

The discovery tests were designed to evaluate the potential of finding resources in the Swarm network. These tests aimed to gauge the efficiency and effectiveness of the discovery process in identifying and accessing resources distributed across the network.

Conducting discovery tests posed unique challenges as the existing testbeds in related works were tailored to scenarios where sensors possessed prior knowledge of each other or communicated through a central node. However, the Swarm requires a more decentralized approach, where such assumptions were not feasible, prompting the need for a new testbed that offered enhanced flexibility. To accommodate this scenario, tests were conducted using our tailored-made framework, the Edge-computing IoT Testing framework, further described in Section 4.

3.2.1 Caching Performance Test

The goal is to measure and compare if the discovery is caching the correct endpoint for the next connection.

Testbed Setup

The setup utilizes a virtualized testbed designed to ensure that devices operate without being interlinked within a singular network. Within this network structure, there's one primary consumer and multiple providers. The test aims to determine if a consumer can effectively memoize the correct endpoint and subsequently use it for a second request.

Performance Metrics and Data Analysis

Performance metrics will be a comparison of the time it takes to establish a successful transaction before and after the implementation of a caching mechanism. This caching mechanism memoizes the endpoint that worked during the last transaction, thus circumventing the need to attempt every endpoint sequentially each time a new transaction is initiated.

If work to note that caching will not improve the first request, but subsequently requests that can happen if the consumer chooses to use the same provider.

The data analysis takes into consideration the position of the first functioning endpoint in a service list and the corresponding time taken to resolve the endpoint. The findings clearly illustrate the increased operational efficiency and time savings achieved with the incorporation of caching, solidifying its value in streamlining transactions within SwarmOS.

Experimental Design

Using the proposed testbed framework, this network can establish a controlled environment, considering various network configurations and topologies. The discovery system under evaluation would be deployed in this testbed, and data collection would commence with various discovery tests conducted under different scenarios and load conditions.

In this study, diverse devices, represented as Docker containers, were established and allocated across various networks within the testbed. The detailed explanation of the SwarmOS discovery algorithm is later provided in Section 5.2. Subsequently, the devices were tasked with identifying a list of agents of type "SwarmCamera", and the resulting log traces were meticulously examined, leading to the identification and resolution of any encountered bugs.

Scalability Evaluation

The discovery phase is identified as a significant portion of the transaction duration. To evaluate the impact of caching on this process, we will measure and compare the

time taken to identify the correct endpoint both before and after the implementation of caching. This comparison will provide insights into the efficiency gains achieved through caching in the system's endpoint discovery.

Ethical and Security Considerations

All data, devices and networks were 100% simulated and adhered to ethical standards throughout the experimentation.

3.2.2 Network Knowledge Enhancement Test

The goal is to determine if a device, after making several requests within the P2P network, enhances its knowledge of the network and can discover devices it failed to find previously. Essentially, it aims to understand whether interactions within the network help a device to increase its awareness of other nodes, leading to more comprehensive network connectivity and understanding.

Testbed Setup

For the Network Knowledge Enhancement Test, the environment constitutes a Swarm network with a predetermined number of nodes, each with varying connectivity levels. Some nodes are intentionally designed to be elusive during discovery. The primary device under scrutiny, termed as the test device, will gauge the expansion of its network awareness.

Experimental Design

Initial Discovery Phase: With no pre-existing knowledge or stored data, initiate the test device's network discovery.

Interaction Phase: Initiate transactions to discover new agents within the same network and store this interaction data in their local cache.

Enhanced Discovery Phase: Verify if, after gaining knowledge about more agents beyond its own network, subsequent searches yield improved results.

Performance Metrics and Data Analysis

Key metrics include the tally of nodes identified during the initial phase versus the enhanced phase, along with a compilation of newfound nodes during the enhanced discovery. Data assessment involves contrasting the initial and enhanced lists of discovered nodes, emphasizing any novel nodes found exclusively in the enhanced stage.

Scalability Evaluation

The expected outcome anticipates that with an increase in network knowledge through prior interactions, the device will demonstrate a clear ability to identify a higher number of nodes, especially those that were previously challenging to detect, during the enhanced discovery phase compared to its initial attempt.

Ethical and Security Considerations

All data, devices, and networks used were entirely simulated, with strict adherence to ethical standards throughout the research. Moreover, we meticulously carried out the testing phase to ensure safety and privacy.

Limitations

Despite the constraints posed by the limited number of devices, the network discovery test remains a valuable tool in our evaluation toolkit. The inherent limitation in device quantity does not negate the test's effectiveness. The test actively contributes to our acceptance testing process by identifying and flagging areas that may require enhancements. This proactive approach leads us to possible limitations in network knowledge, ultimately advancing to a more refined and robust system.

3.2.3 Query Discovery vs Cache-Assisted Discovery Time Test

The goal is to measure and compare the time taken for two nodes to establish a connection using a query and when they have prior knowledge of each other.

Testbed Setup

A Swarm virtual network equipped with multiple agents.

Test Nodes: Two specific nodes, Node A and Node B, selected for assessing connection establishment.

Experimental Design

In the Query-Assisted Connection Phase, start Node A and Node B, ensuring they have no prior information about each other. From Node A, send a query to the network and make sure Node B responds to it. Then, record the duration from the start of the request to the end of the whole transaction.

In the Cache-Assisted Connection Phase, make sure Node A and Node B already know each other's network details from previous interactions. Then, have Node A initiate

a direct connection to Node B. Finally, calculate the duration it takes from the beginning of this direct request until the connection is successfully established.

Performance Metrics and Data Analysis

Metrics:

- Duration for cache-assisted connection.
- Duration for query-assisted connection.

Data Analysis: Evaluate the connection times from the two phases to confirm the faster method and its margin of efficiency.

Scalability Evaluation

Query Discovery typically requires more time and resources as it involves actively seeking out nodes in the network for every request, which can become inefficient as the network grows. On the other hand, Cache-Assisted Discovery leverages stored information about previously discovered nodes, potentially offering quicker and more efficient discovery processes, especially in larger networks.

Ethical and Security Considerations

All data, devices, and networks used were entirely simulated, with strict adherence to ethical standards throughout the research. Moreover, we meticulously carried out the testing phase to ensure safety and privacy.

3.3 Fairness

This test was designed to evaluate if the reward system is effective to achieve fairness in the network, i.e. good service providers and service consumers are rewarded and the malfunctioning or malicious ones deterred from interacting with peers in the network.

3.3.1 Reputation Test

Testbed Setup

An analysis of a reputation system in a simulation can provide valuable insights into how such systems function in a controlled environment and identify any potential strengths or weaknesses. By testing the reputation system in a simulated setting, we

can analyze its behavior under various scenarios, such as different levels of cooperation, different types of transactions, and different numbers of participants. This can provide a deeper understanding of the interplay between individual behavior and the reputation system and allow for a systematic evaluation of the system's effectiveness.

Using the testbed proposed in this work, multiple test cases encompassed various types of bad actors within a network. These bad actors, acting individually or collaboratively, sought to gain unfair advantages over other nodes or the entire network. Through comprehensive experimentation, we closely observed the network dynamics, analyzing how these bad actors affected the overall performance and security.

The test simulates both providers and consumers, utilizing a machine equipped with an Intel(R) Core(TM) i7-3632QM CPU @ 2.20GHz and 16GB of RAM. To maintain consistency, a ratio of 80% devices acting as providers and 20% devices as consumers will be employed throughout the testing. It is important to note that the chosen ratio was selected in an arbitrary manner for the purpose of the experiment.

The test was conducted with a total of 41 simulated devices, comprising 32 providers, 8 consumers, and 1 blockchain payment system.

Performance Metrics and Data Analysis

The performance metrics used is the number of interactions among undesirable peers with regular peers until the undesirable peers are deterred from interactions in the network.

The Data Analysis is done generating graphics with the number of interactions in the x axis and the reputation of actors in the y axis. Lines with the reputation mean of each kind of actor are presented. A threshold line is also presented showing the exclusion of the undesired actors from the network.

Experimental Design

With the testbed proposed in this work, 6 test cases will be simulated with different types of undesirable actors to create a robust analysis of a reputation system in a simulation. These test cases vary the proportion of bad actors and good actors to evaluate the network's ability to detect and respond to adversarial behavior, thus enhancing its robustness and trustworthiness.

The identification of particular malicious actors was built upon the work (GUO; CHEN; TSAI, 2017). Furthermore, in addition to malicious actors, the actors with low quality of service were also considered.

The use cases are:

1. **Providers with Invalid HTTP responses**: this use case is composed by regular Consumers, and Providers that are part regular, and part providing *invalid HTTP responses*. This Provider with *invalid HTTP responses* represents actors that are malfunctioning or that are fraudsters, i.e. they receive the payment but do not provide the service. Servers initially operated correctly but experienced crashes and generated erroneous HTTP responses after five requests

2. **Providers with slow response time**: this use case is composed of regular Consumers, and Providers that are part regular, and part providing *slow response time*. This provider with *slow response time* represents a peer with poor quality of service due to processing or communication limitations. Servers initially operated correctly, but after five requests, started to present slow response times.

3. **Providers doing on-off attack**: this use case is composed of regular Consumers, and Providers that are part regular, and part are malicious by providing *on-off* attack. This provider is modeled by answering invalid HTTP responses randomly, representing a provider that sometimes does not provide the service after receiving the payment.

4. **Providers doing opportunistic on-off attack**: this use case is composed of regular Consumers, and Providers that are part regular, and part are malicious by providing *on-off* attack. This provider is modeled by answering invalid HTTP responses, until its reputation is near to the selection threshold, then it starts offering good quality of service to improve its reputation, operating with the reputation above, but close to the selection threshold.

5. **Gang attack**: this attack is composed of both providers and consumers. Consumers do self-promoting attacks, that are represented by giving the highest reputation points to the providers that are part of the gang independently of having a service actually provided. Consumers also do the bad-mouthing attacks, that is implemented as providing bad evaluation of providers outside the gang independently of the quality of service provided. The providers part of the gang do fraudster attack, that is receiving the payment without providing the service. This way, they try to keep a good reputation by the evaluations of the consumers that are part of the gang.

Table 3: Details of the participants of the experiment

Use Case	Type	Description	Provider	Consumer
1	Malfunctioning Provider	Providers with Invalid HTTP responses	Invalid HTTP	-
2		Providers with slow response time	Slow response time	-
3	Malicious Provider	Providers doing on-off attack	On-off attack	-
4		Providers doing opportunistic on-off attack	Opportunistic attack	-
5	Gang Attack	Gang attack	Fraudster attack	Self-promoting attack + Bad-mouthing attack
6		Opportunistic gang attack	Opportunistic on-off attacks	Self-promoting attack + Bad-mouthing attack

6. **Opportunistic gang attack**: this attack is composed of both providers and consumers. Consumers do self-promoting attacks, that are represented by giving the highest reputation points to the providers that are part of the gang independently of having a service actually provided. Consumers also do bad-mouthing attacks, that are implemented as providing bad evaluation of Providers outside the gang independently of the quality of service provided. The Providers part of the gang do on-off attacks, This provider is modeled by answering invalid HTTP responses, until its reputation is near to the selection threshold, then it starts offering good quality of service to improve its reputation, operating with the reputation above, but close to the selection threshold.

The use cases are outlined on Table 3.

Tests were conducted with agents with undesirable behavior in varying quantities, specifically using 12.5%, 25%, and 50% of providers and/or consumers, ensuring a comprehensive analysis of the reputation system's performance.

The experiment involved a single miner, 8 consumers, and 32 cameras. All running in a single network inside the **Edge-computing IoT Testing framework**. The choice of a single network was deliberate, as the objective was to assess fairness rather than testing connector or discovery mechanisms. A namespace was defined to facilitate understanding each camera's role, where camera1 to camera32 represented well-behaved cameras, while camera33 and beyond simulated malfunctioning or malicious cameras.

The data collected was the whole reputation blockchain, where we can plot the reputation of each camera service on the block number, where block number is the chronological order of the block's inclusion in the reputation blockchain. By analyzing these values, we

can see if consumers stop using malfunctioning providers.

Scalability Evaluation

This test identifies the number of interactions between undesirable peers with regular peers until the undesirable peers are deterred from interactions in the network. The scalability analysis extrapolates the results doing a theoretical analysis of its implications in a big global network of billions of devices.

Ethical and Security Considerations

Ethical and security considerations were not applicable in this study, as all data was simulated and confined to a single computer. Consequently, the data exchanged during the experiment was entirely fabricated and presented no real-world implications or risks.

Limitations

This work addresses only specific types of attacks against the network, and it is essential to acknowledge that new types of attacks emerge unpredictably in the future. As the threat landscape continuously evolves, further research and proactive measures are imperative to counter potential novel attack vectors effectively.

This limitation, rather than detracting from the validity of the test, highlights the forward-thinking nature of the study. The ongoing evolution of the threat landscape necessitates a flexible testbed paired with continuous research and proactive measures, emphasizing the relevance and timeliness of this work in addressing current and potential future attack vectors.

3.4 Privacy

The privacy tests were designed to evaluate the device's potential to govern interactions in a decentralized and autonomously way, inside the Swarm network. These tests aimed to gauge the efficiency and effectiveness of the authentication process in evaluating and allowing resources distributed across the network.

This limitation, rather than detracting from the validity of the test, highlights the forward-thinking nature of the study. The ongoing evolution of the threat landscape necessitates a flexible testbed paired with continuous research and proactive measures, emphasizing the relevance and timeliness of this work in addressing current and potential future attack vectors.

3.4.1 SmartABAC Performance Test

Testbed Setup

SwarmOS incorporates a dedicated privacy module that integrates an ABAC system and a SSI framework. The ABAC system allows for fine-grained control over access to resources based on attributes, while the SSI empowers users to control their own identity data. As part of our testing process, we indirectly assess the privacy module's functionality and effectiveness while conducting various other tests on SwarmOS. These tests indirectly evaluate how the privacy module performs under different scenarios and interactions within the SwarmOS ecosystem.

Performance Metrics and Data Analysis

The study aimed to assess the execution time among different ABAC models, including open-source models like Policy Machine (PM), Hierarchical Group and Attribute-Based Access Control (HGABAC), and Extensible Access Control Markup Language (XACML). The comparison involved evaluating Time to Evaluate Policies, Policy Size, and Lines of Code.

Additionally, the second test focused on the performance of two SmartABAC implementations across four distinct devices. The results demonstrated the functionality of SmartABAC on different platforms.

Experimental Design

In the initial phase of the test, where we compared SmartABAC with other open-source models, we gauged the total time, in milliseconds, required to evaluate a single request against three different policies, repeated 3000 times. These assessments were conducted on a laptop equipped with a 1.8-GHz quad-core CPU and 8 GB of RAM. Additionally, we conducted a comparative analysis of policy sizes and lines of code for the implementation of these policies.

Moving on to the second part of the test, which focused on the performance of SmartABAC across various platforms, we considered a spectrum of devices. These ranged from non-constrained devices like a commercial laptop and a Labrador (a single-board computer) to constrained devices such as the ESP32 and Pulga, consisting of microcontroller units (MCUs) running bare-metal software.

Scalability Evaluation

This testing approach allows us to discern whether the policy system might become

a bottleneck in the system's scalability. By repeatedly assessing the time required to evaluate requests, the study aims to gauge how well the policy system scales as the transaction volume increases. This measurement becomes crucial in scenarios where the policy system's efficiency directly influences the speed and responsiveness of the entire transaction process.

Ethical and Security Considerations

All information, devices, and networks employed were completely simulated, and the experiment adhered rigorously to ethical standards at all stages.

Limitations

This study focuses on a specific aspect of the process rather than conducting a comprehensive integration test of the entire transaction. It provides a preliminary insight into the system's functionality but acknowledges the necessity for additional tests to validate scalability under more realistic deployment conditions. The focus on a particular aspect allows for a detailed examination, providing valuable insights into how enhancements in one part of the system can positively impact the entire process.

4 EDGE-COMPUTING IOT TESTING FRAMEWORK

As mentioned before, a new testbed framework was proposed because other works had frameworks tailored for a single use case, architecture or protocol. A new flexible testbed was proposed to cater to the diverse and ever-expanding landscape of IoT applications. The proposed testbed framework is designed to be versatile and adaptable, capable of accommodating different use cases, network architectures, and communication protocols within the IoT domain.

This chapter presents the proposed approach to create a general framework for testing, working as an intermediate step between idealization and physical tests. This framework is designed to be as flexible as possible, and to be open to measure various parameters of the system such as performance, realism, ease of deployment, and scalability. The next step would be to implement several test cases for testing the scalability of the swarm computing-based system. These techniques will be chosen based on their relevance to the swarm computing domain and their ability to measure the scalability of the system.

The implementation and testing of these techniques will be carried out within the testbed framework developed in the first step. This approach will help to provide a comprehensive understanding of the scalability of the **Edge-computing IoT testing framework** and will help to identify the strengths and weaknesses of the system. The results of the testing will be analyzed and presented in a structured manner, with recommendations for further improvements to the system. This approach will provide a solid foundation for improving the scalability of swarm computing-based systems.

4.1 Docker/Virtualization

Virtualization creates a virtual machine that runs on top of the physical hardware, which provides isolation and compatibility for different operating systems. Containers, on the other hand, are a lighter-weight method of virtualization that allows multiple isolated

applications to run on a single host machine, sharing the underlying operating system. The main advantage of containerization is that it reduces overhead, allows for better resource utilization, and makes it easier to deploy, scale, and manage applications. Additionally, containers provide a portable and consistent environment for applications, making it easier to move applications between different environments, such as development, testing, and production.

Docker is a lightweight virtualization tool that can completely encapsulate an application and its dependencies within a virtual container. The application provides a control interface between the host OS and the application's containers. Many research efforts are being directed to joining the Docker advantages with IoT (KAO, 2020).

Containers can communicate between themselves using the Docker networks, a Docker network uses a software bridge that allows containers connected to the same bridge network to communicate while providing isolation from containers that are not connected to that bridge network, so the developer has control over which containers are connected and how they are connected, creating the topology as flexible as wanted.

In this work, Docker was selected as the preferred technology due to its well-established track record, flexibility, and extensive array of add-ons tailored to support various network architectures, communication protocols, and measurement software. The decision to employ Docker as the underlying framework was driven by its proven reliability and the versatility it offers, allowing for seamless adaptation to diverse IoT scenarios and facilitating efficient experimentation with different configurations and tools.

A strategic choice was made to incorporate multiple applications within a single container, this decision was taken for two reasons. Firstly, this approach was adopted to assess the resource competition that arises when encapsulating various microservices within a single IoT device. Secondly, the use of Docker runtime options facilitated the emulation of device CPU and memory constraints, allowing for comprehensive simulations and evaluations of resource limitations that occur in real-world IoT environments. (RUNTIME..., 2021).

Furthermore, to facilitate multi-Docker environments, Docker-compose serves as a valuable tool for constructing and executing multi-container Docker applications. By enabling users to define an application's services, networks, and volumes in a single file, known as a docker-compose.yml file, this tool streamlines the process of creating and starting all specified services with a single command, thus minimizing the time and effort required to manage intricate applications. Consequently, the utilization of Docker

Table 4: Characteristics of testbeds methods

	Fidelity	**Cost**	**Setup Time**
Physical	High	High	High
Emulated	Medium	Medium	Medium
Simulated	Low	Low	Low

in conjunction with Docker-compose offers the advantage of efficiently running a group of containers, simplifying the tasks of container creation, removal, and updates, thereby enhancing the scalability of the testing process for a greater number of devices and applications.

4.2 Testbed

Considerable research efforts are currently dedicated to the testing of emerging IoT architectures. Such research can be categorized into three primary methods, **physical**, **emulated**, and **simulated testbeds**. Table 4 compares testbeds based on fidelity, cost and setup time.

Physical testbeds use real devices to do experimentation-based prototyping. The idea is to create a platform where we connect these devices like we were connecting in the real world and use them to test if the real platform will work. This type of testbed provides a practical and realistic simulation of how these devices and technologies would work in a real-world scenario, allowing researchers and developers to test their theories and ideas in a controlled environment. They are essential for evaluating and optimizing the performance of IoT systems and can be used to identify and resolve any issues that arise during the development and deployment process.

Emulated testbeds use an emulator to mimic the hardware behavior. Emulators are software tools designed to mimic all the essential hardware features of actual devices, thereby facilitating the simulation of one system's behavior at the hardware level to resemble that of other devices during testing. This type of testbed provides a controlled and isolated environment for experimentation, without the need for expensive and complex physical hardware.

Simulated testbeds employ device models to replicate the behaviors of complete systems, wherein one or more models of devices are employed to specifically target particular aspects of the platform. This testing approach involves the creation of computer models and algorithms that simulate the behavior and interactions of devices, networks, and

protocols, mirroring real-world scenarios. By utilizing such simulated environments, researchers can systematically evaluate and analyze the performance and functionalities of IoT architectures, without the need for physical implementation, enabling efficient and controlled experimentation in a virtual yet realistic setting.

4.2.1 Physical Testbeds

Physical testbeds use real devices to perform tests. They present the advantage of a highly accurate view of the system depending on the number of devices used and they provide a degree of realism that simulations cannot achieve (NORDSTRÖM et al., 2007). But the drawbacks are the cost and difficulty to set up a real IoT testbed, and with the quick evolution of IoT hardware, it's safe to assume that they will be outdated fast. These drawbacks can be lessened with the use of open-testbeds, they are readily available and deployed hardware and networks for experimentation, if you wanna use an open testbed, you have to check some characteristic criteria to choose with a testbed to use (CHERNYSHEV et al., 2018), below are some example criteria.

1. *Scale* – Maximum size of devices supported by the testbed.
2. *Environment Type* – Practical applicability (smart city, structural monitoring, low-level protocols, etc.).
3. *Communication Protocol* – Which protocols are supported by the testbed.
4. *Mobility* – If mobility is tested too on the testbed.

Several large-scale testbeds have been deployed in recent years, many with a specific focus. Some examples are the FIT IoT-LAB (ADJIH et al., 2015) in low-level protocols, SmartSantander (SANCHEZ et al., 2014) in smart cities, Web of Things TestBed (WoTT) (BELLI et al., 2015) in Web of Things, JOSE (TERANISHI et al., 2016) in structural monitoring and many others.

4.2.2 Emulated Testbeds

Emulators are a middle ground between physical and simulated testbeds. The experimenter can use real code to emulate the behavior of a device or the whole network while emulating the hardware behavior too. The drawbacks are that you need an emulator for each type of device, being harder and harder to test in such a heterogeneous environment.

Many emulators are being developed, for many OSs. Cooja (ÖSTERLIND, 2006) is an extensible Java-based emulator developed for ContikiOS devices. The code to be executed by the node is exactly the same as you upload to physical nodes.

MAMMotH (LOOGA et al., 2012) is a large-scale IoT emulator, is able to emulate thousands of devices per Virtual Machine, whose architecture presumes three distinct scenarios, namely:

- mobile devices connected via GPRS to a base station forming a star topology
- a stand-alone WSN connected to a base station via GPRS
- constrained devices (e.g. sensors) connect to proxies, which in turn connect to the backend

4.2.3 Simulated Testbeds

Between all types of tests, simulated testbeds are the best scaling, the models are simplified versions of devices, so usually you can run many simulated devices inside a machine. The drawbacks are that the test is as good as the models' fidelity with reality, and to simulate a heterogeneous system, you will need many models, one for each different application. Dias et al. (2018) makes a concise overview of test approaches and tools including many with simulated capabilities. The disadvantage of this type of testing is that simulations are useful for problems with high-fidelity models, as the nature of an IoT problem is dynamic and heterogeneous. Developing models that faithfully represent the behavior of a system can be an arduous task.

Chernyshev et al. (2018) identified three main categories of simulator used in IoT testbeds, full-stack simulators, focused end-to-end support to all IoT elements, data processing simulators, focused on simulating the data flow and processing, and network simulators focused on simulating the network and its characteristics. Like open-testbeds, we should look at some characteristic criteria to choose which testbed is better suited for testing, like scope, type, language, architecture, etc.

Some examples of testbed simulators include CupCarbon (MEHDI et al., 2014), Cooja (OSTERLIND et al., 2006), OMNeT++ (VARGA; HORNIG, 2008), NS-3 (HENDERSON et al., 2008), and QualNet (QUALNET, 2021).

OS-level virtualization testbed uses the same software that runs on the real deployment hardware thus offering the chance to test errors that happen because of software

bugs and problems induced by errors in the network like packet loss/corruption, network partition, etc. The drawback of these tests is the cost of computational processing of virtualizing a whole device.

It can provide a step between simulated and physical testbeds. While simulated can scale with fewer resources, virtualization can be faster and more realistic since it is not needed to create a model for each one of the applications being tested, and since we can virtualize many devices on a single machine, it can scale better than physical testbeds.

There is a lot of research being done in the virtualized testbed space. Dockemu (TO; CANO; BIBA, 2015) is a simulator that joins Docker and NS-3 (a network simulator), the framework facilitates creating new containers and connecting them using NS-3 to test simulated networks. NS-3 offers ways to create more complex networks, but at the time of writing, the Dockeremu framework only has 2 modes, all nodes are connected wired or wireless, and there is no template for more complex networks, another drawback is that the tool does not test the code, checking for possible errors in different message orders, disconnection, network splits and the tool is limited on the maximum number of containers the host machine can virtualize.

UiTiOt (LY-TRONG et al., 2018) is a hybrid system that mixes virtual wireless nodes with real physical devices to perform a variety of IoT experiments. The tests are focused on testing multi-hop routing on wired sensor networks (WSN), the framework can test different architectures (mesh, tree, etc...), but the framework does not do tests where agents are connected directly to the Internet.

EMU-IoT (RAMPRASAD et al., 2019) is a platform focused on the architecture of producer→gateway↔application where it has a well-defined data pipeline, but this architecture does not reach more decentralized IoT architectures where cascading failures can occur with different orders of messages, messages dropped, errors in connection.

ELIoT (MäKINEN; JIMéNEZ; MORABITO, 2017) is a platform that enables the emulation of simple IoT devices on Docker. The author uses CoAP + LWM2M for communication and in the work, they test the scalability of the platform using the time that each docker takes to connect to a single LWM2M Server. This testbed is focused on LWM2M connections.

None of these solutions solves our problem, these platforms are focused on testing the network, usually, testbeds are developed with a specific use case in mind, and adapting these tools to work in different use cases can be laborious.

Since the goal of this work is to create an intermediate step between idealization and full deployment, the simulated testbeds seem like a good fit, with great scalability and low cost we can start fixing bugs and creating testing environments really fast. Also, it's easier to manipulate the simulated environment, which can be useful for debugging and verifying the functionality of the system.

The only problem is that the realism of a simulation depends on the realism of the model. The way this work improves the realism of the model is by using containers, where we can run the same code as the one that will be deployed in production. Using the same code brings two big advantages. This consistency in code execution helps to ensure that tests accurately reflect real-world scenarios and reduces the risk of unexpected behavior due to differences between test and production environments. Another advantage of using containers is that it can improve the realism of tests. By running simulations in a containerized environment, the testbed can provide a more accurate representation of the system under test and help to identify potential issues before they occur in the production environment.

4.3 Testbed Framework Design

The key idea of this section is to propose a general framework for testing IoT platforms. The testbed is based on OS-level virtualization, which means that we can run (at the expense of hardware usage) the same software executed in deployed devices, testing for software bugs, errors in interoperability, monitoring CPU/memory usage, etc.

Since IoT devices are so heterogeneous, one of the framework goals is to be as general as possible, without making any assumptions on the architecture of the network. Many of the state-of-the-art testing frameworks are built with some architecture/usage in mind, making it harder (or even in some cases impossible) to test projects with different use cases.

Our emphasis lies in creating a general, realistic, and reproducible testbed, in other words, strive to minimize the error between simulation and real test and aim to have consistent results without losing any comprehensiveness. It's worth saying that reproducibility does not mean that the results will be exactly the same in different runs, just that the results are consistent between runs. To help us with this framework, we will need some tools:

- OS-OS-level virtualization, to create isolated user spaces on a single device, increas-

ing the number of applications executed without increasing too much the cost of hardware.

- Network emulation, since virtualization gives us an unrealistic network, we need an emulation tool to emulate a more realistic (with delay, corruption, lost packets) network.

- Measurement tools, to measure and monitor desired metrics.

4.3.1 Strategy

Emulation can have more fidelity, but it is known to have poor efficiency (WHITE; PILBEAM, 2010). Since virtualization is shown to introduce negligible overhead in terms of resource consumption (MORABITO, 2017; MORABITO et al., 2017), we choose to use a testbed based on virtualized devices, which also means that we can run the same software as the one deployed on physical devices, this approach offers the advantage of identifying potential bugs and issues within the code, contributing to improved software reliability. The virtualization drawback is that there are no tests on the device hardware architecture when compared with an emulation. And when compared to physical testbeds, the network being tested is simulated and the hardware access has to be mocked up.

Fortier & Michel (2003) divides a testbed into three components: (1) an **experimental subsystem**, (2) a **monitoring subsystem**, and (3) a **simulation-stimulation subsystem**. The experimental subsystem is the collection of models and/or prototypes we wish to test. It defines the structure and characteristics of the network, how many devices are tested, how the devices are connected, the limitations (if any) of each device, etc. The monitoring subsystem is the set of tools used to extract raw data and collect information about the experimental subsystem. It also comprehends the tools that help organize and analyze the data collected, used to measure the target characteristics of the test case. The simulation-stimulation subsystem can help create the initial conditions and provides the handles and knobs necessary for the experimenter to input events into the experimentation environment. Our framework architecture is based on Fortier & Michel (2003) subsystems: **experimental**, **monitoring**, and **simulation-stimulation**.

The **experimental subsystem** is structured in agents, devices, and networks. An agent is a computer program characterized by reacting to the environment and inputs, having autonomy, goal orientation, and persistence (FRANKLIN; GRAESSER, 1996). In the Swarm Computing paradigm, each agent corresponds to a service provider or consumer

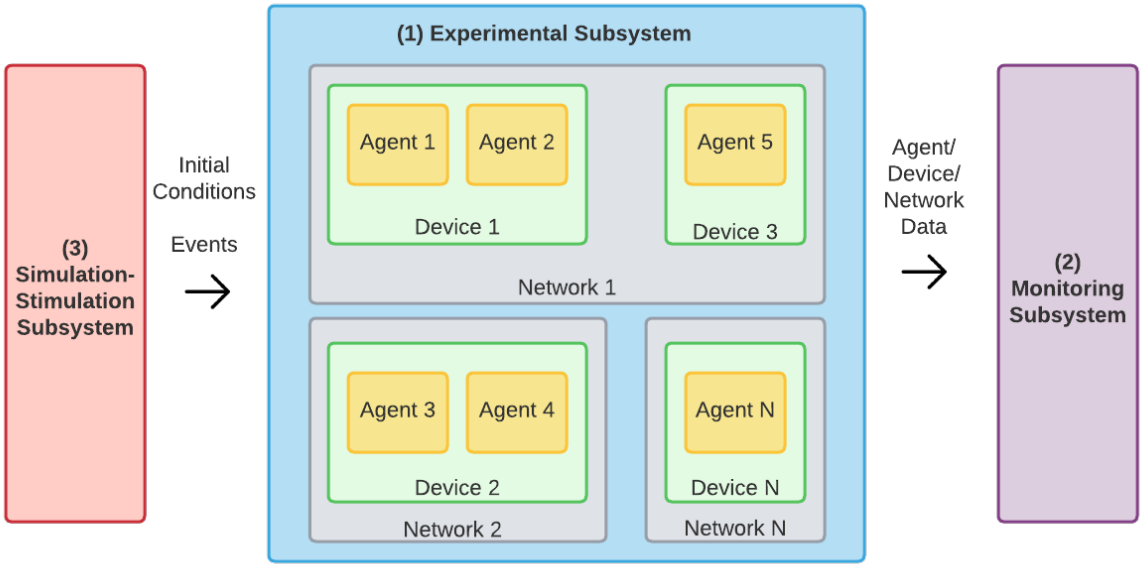
(COSTA et al., 2015a). A device is a unit of physically independent electronic equipment used to compute or support agents. The network defines the set of devices connected that can share resources directly between themselves. The network also defines the mechanisms used, isolating or opening communication between containers.

On the experimental subsystem, one network can have one or more devices that can have one or more agents. Also, the job of the experimental subsystem is to create initial conditions providing information about their devices such as resource constraints (CPU, memory, throughput, etc.). Likewise, defining properties of the networks (delay, packet loss, traffic rate limit, etc.), initial environment variables of each agent, and others.

The **simulation-stimulation subsystem** is composed of tools and knobs to provide real-world system inputs. It provides the experimenter the ability to create scenarios to test parts of the experimental subsystem. It can also pause or delete some agents, devices, or networks.

The **monitoring subsystem consists** of tools to extract metrics and analyze the collected information. It can monitor target data from any agent, device, or network, save all packets trafficked in a network, and also help with the organization and automatically create a graphic representation of the data collected. Some examples of useful metrics that can be extracted are memory usage, CPU load, number of CPU throttled period intervals, number of I/Os currently in progress, bandwidth metrics, requests delays, and packet data. Figure 4 displays the testbed design structure.

Figure 4: Proposed testbed structure, each network can have many devices which can have many agents/applications



4.4 Testbed Framework Implementation

To give a concrete implementation of the proposed framework, we choose to use well-known off-the-shelf tools, the reason is that usually, off-the-shelf software has thousands of work hours on developing, testing, making it more reliable and generic than custom-made software, giving us a way to create a more generic framework. These tools also provide a cost-effective and efficient solution. These tools are widely available, customizable, easy to use, and have many extensions making them ideal for evaluating many architectures.

As mentioned above, the design of the testbed is based on the Fortier & Michel (2003) subsystems. The following text will delve into the specifics of each software selected for the foundational version of the proposed testbed.

4.4.1 Experimental Subsystem

A key feature to implement our virtualization testbed is the container manager. We selected Docker since it is well-documented, compatible with most modern operating systems, and widely used in many research works (KAO, 2020). Docker is a lightweight virtualization tool that enables the user to completely encapsulate an application and its dependencies within an OS-level virtualization, these encapsulations are called containers. One of the advantages of using docker as a virtualization tool is that we can execute the same code as the one running on the physical device, being able to test for other aspects of the code, like security, performance, interoperability, robustness, conformance, compatibility or multi-objective testing.

A container is a virtualization paradigm (RAMALHO; NETO, 2016) in which the operating system kernel allows the existence of multiple isolated user-space instances, which is system memory allocated to running applications with their configuration files, library, and software isolated from one another. User space instances have a lower level of privilege than the operating system kernel.

Containers use fewer resources than Virtual Machines (VMs) because they share the host operating system (POTDAR et al., 2020; RAMALHO; NETO, 2016). We are using containers as tools to be used for testing in IoT, virtualizing devices, and making it easier to scale and run cheaper tests since a single machine can virtualize many devices. Many research efforts are being directed to leverage the advantages of using Docker in the IoT space (KAO, 2020).

In our framework implementation, each device is represented by a container. This

decision was taken for two reasons: to evaluate resource competition caused by wrapping different microservices into one device; to simulate the device CPU/memory constraints using Docker runtime options (RUNTIME..., 2021).

Containers can communicate between themselves using Docker networks. A Docker network[1] uses a software bridge that allows containers connected to the same bridge network to communicate while providing isolation from containers that are not connected to that bridge network, so the developer has control over which containers are connected and how they are connected, creating the topology as flexible as wanted.

One thing to keep in mind, the number of containers run by a single computer is still limited by the resources available on the host machine and by software, for example, the maximum number of containers per Linux Network Bridge is 1023 (Seetharami Seelam, 2023). So, to connect more containers we need to use more bridges.

Together with Docker, two tools are used: Docker-compose and Pumba[2]. Docker-compose is a tool for creating and running multi-container Docker applications. With a configuration file, we can create/edit/remove containers from the environment. So, another advantage of using Docker with Docker-compose is to run a group of containers together simplifying the work of creating, removing, and updating containers, making it easier to escalate the number of devices/applications being tested just by adding some lines of code. Listing 1 shows a Docker-compose configuration file, we can see 3 different containers (payment, consumer, and camera) connected in 2 different networks, also we can see resource constraints being applied in the level of devices (containers).

We can create/delete/update entire networks in minutes, even with heterogeneous agents. We can also define many networks, connecting them and creating complex topologies, each network can have custom network drivers and options, increasing the control over characteristics that the experimenter can use. We only need to configure it once, after this we can mix and match any type of topology we want just by changing the docker-compose configuration.

One characteristic of the docker network is that it creates an unrealistic network, without delays, latency, and packet loss. To apply the expected network characteristics of the deployment environment, Pumba, a chaos testing command-line tool for Docker containers was utilized. Pumba can emulate wide-area network failures (adding network delay, packet loss, corruption, duplicated packets, etc.) and stress-testing container re-

[1]https://docs.docker.com/network/
[2]https://github.com/alexei-led/pumba

```yaml
version: "3"
networks:
  network1:
  network2:

services:
  payment1:
      build:
        context: ./project
        dockerfile: DockerfilePayment
    networks:
      - network1
      - network2
  consumer1:
      build:
        context: ./project
        dockerfile: DockerfileConsumer
    networks:
      - network1
  camera1:
    build:
      context: ./project
      dockerfile: DockerfileCamera
    networks:
      - network2
    deploy:
      resources:
        # (optional) Resource constraints
        # on each container
        limits:
          cpus: '0.5'
          memory: 100M
```

Listing 1: Example from Docker-compose, a configuration with 2 networks and 3 devices

sources (CPU, memory, I/O, and others). Besides that, Pumba can kill, pause or stop containers, useful if we need to test failures in the network.

The experimental subsystem can be adjusted using the docker configurations, making it possible to:

- Change the number of devices, having different runtime options, limiting the CPU and memory that each container will have access;
- Use a network emulation tool, to emulate the properties of wide-area networks like network delay, throughput, packet loss/corruption/duplication;
- Create/Delete/Lock some devices in the middle of transactions to analyze the behavior of single devices or the whole network;
- Change the topology using docker-compose networking feature, creating many networks, connecting them, and having different characteristics in each one of them.

For this work, we used Docker-compose and Pumba to create initial conditions on containers and networks respectively.

4.4.2 Simulation-Stimulation Subsystem

As mentioned before, the simulation-stimulation subsystem is composed of tools and hooks to provide real inputs. Many tools can be used to form this subsystem, some examples are stress testing tools to create high loads for agents, custom scripts to create duplicated/corrupted/incomplete requests, or common requests to test the typical scenario running in the network. For example, in other works, Tsung was also used as a load-generating tool. For this work, we decided to start simple and only start with small cases with common requests, so our simulation-stimulation subsystem consists of standard RESTful requests made in curl[3] requests.

4.4.3 Monitoring Subsystem

The monitoring subsystem is the measurements and data collection of the framework. Despite presenting a broad framework, at first, we decided to apply a more restricted test, selecting 3 metrics for analysis: CPU consumption, memory consumption (to evaluate the testbed), and network degradation adding packed delay (to be used as a test case).

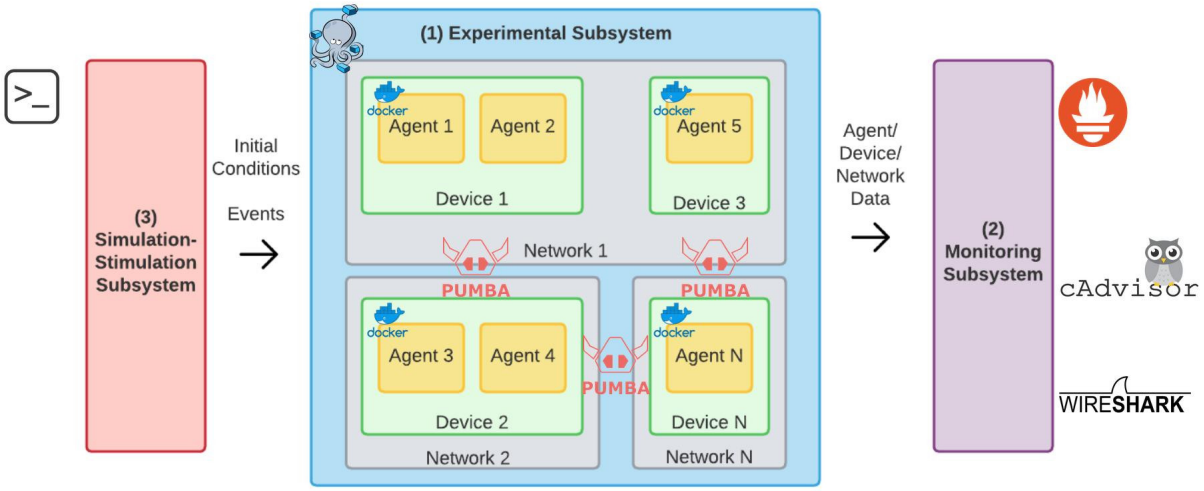
[3]https://github.com/curl/curl

Measuring CPU and memory usage metrics in docker can be tricky. Casalicchio & Perciballi (2017) showed that many monitoring tools measure different metrics. We selected the cAdvisor tool based on the same study which showed that cAdvisor measures the effective workload generated by the container on the CPU, which is the data we are looking for in this case. Together with cAdvisor, there are many plugins to export statistics. We selected Prometheus, a well-known monitoring system, with which it is possible to create graphics and alerts in real time.

For the network degradation test, the Wireshark software was used. Wireshark can save all packets trafficked in the network, making it easier to check for bottlenecks, and periods of great traffic of packets and compare many different cases.

After finishing all subsystems, the testbed structure implemented looks like figure 5.

Figure 5: Testbed structure implemented with the off-the-shelf software



4.5 Testbed Evaluation

Prior to commencing the scalability analysis of SwarmOS, a preliminary evaluation of the testbed was done. The evaluation includes an analysis of CPU overhead to discover whether this testbed introduces substantial overhead that might impede the testing process. Followed by an exploration of the upper limit of devices that the testbed can effectively accommodate.

Using Docker-compose we can create/delete/update entire networks in minutes, we can see on listing 1 how easy it is to create a simple network even with heterogeneous agents.

Having a different network is easy too, we can define many networks with different drives if needed and connect each one of the containers in different networks, building complex networks. In the example, in the listing 1, we can see 3 different containers connected in 2 different networks.

The proposed methodology is evaluated with the use case of a Swarm surveillance application. The consumer together with the broker creates a query. This query is sent across the network seeking cameras within a radius around the issuer and contracts the ones he wants, a payment system will occur and he will receive a token to access the cameras for a fixed amount of time. Then, after using the cameras, a reputation system will happen. All of this is considered one transaction. In this use case, our consumer is looking for cameras around his neighborhood. An example of security cameras found when executing the query is given in Figure 6.

The tests will be focused on a couple of areas. First, how do small problems in the network transform the transaction for all devices, for this, in the network degradation test part we will add a bit of delay in all the networks and check for the change in the time of completion of a standard transaction.

On the Docker CPU overhead test, we want to know if the testbed adds much overhead in the test, and how much this overhead grows with more containers.

And for the last part, we will calculate the maximum value of virtualized devices we can run on a machine and check if it's possible to run tests of "real IoT applications", with hundreds, even thousands of devices.

4.5.1 Docker CPU Overhead Test

In this work, we consider Docker CPU overhead and any processing power not used directly by the application. The Docker overhead is depending on several processes whose values vary depending on the number of containers running, the number of exposed ports, etc.

Running the command pstree (Figure 7), we can see the process tree that Docker creates to run *dockerd* → *containerd* → *containerd-shim* → *application* and *docker-proxies*.

To analyze the CPU consumed by the application and by the system (dockerd + containerd + containerd-shim + docker-proxy), in this work, we did use cAdvisor[4], a tool to inspect resource usage and performance characteristics of running containers, together

[4]https://github.com/google/cadvisor

Figure 6: Example of security cameras found

Figure 7: Output of pstree: Highlighted in red the processes created by docker and in green the processes created by the application

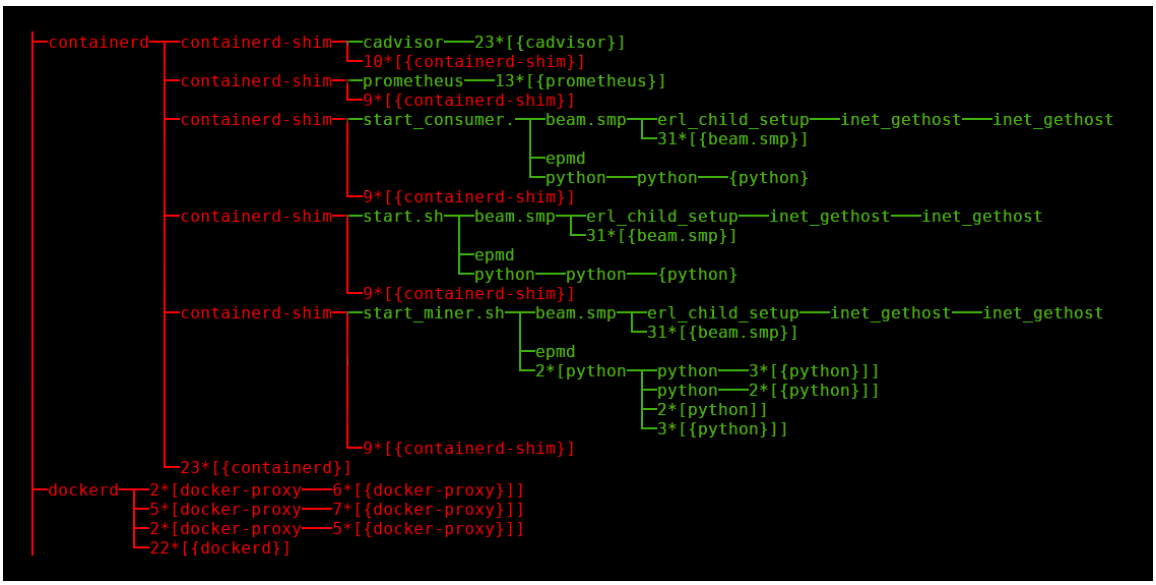


Figure 8: Payment System: Cumulative system CPU time consumed by they system and user in seconds

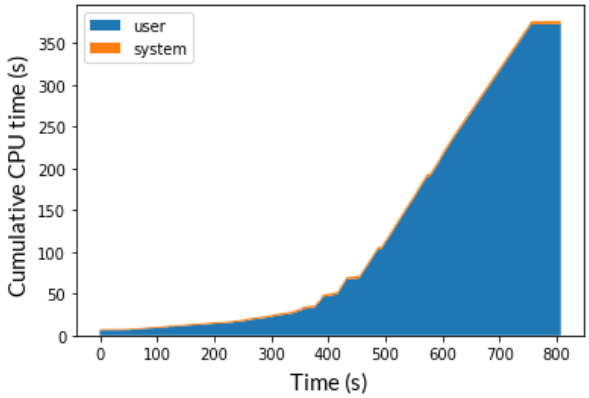
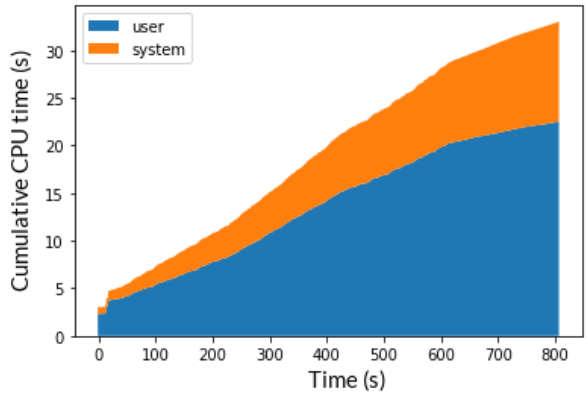


Figure 9: Consumer Application: Cumulative system CPU time consumed by they system and user in seconds



with Prometheus[5], a tool that collects and organize metrics from targets.

Together, both can be easily configured with Docker-compose, since both already have pre-compiled container images. cAdvisor exposes container and hardware statistics and Prometheus collects them giving instant results. The metric used was `container_cpu_system_seconds_total` (cumulative system CPU time consumed) and `container_cpu_user_seconds_total` (cumulative user CPU time consumed). This can be seen in figures 8 and 9. The rate of CPU used by the system varied from 0.97% to 36.77%.

4.5.2 Virtualization Limit Test

To test how much this testbed is scalable, we want to test how many brokers + applications pairs can be run on a single machine and test how much overhead the docker adds to the framework.

[5]https://github.com/prometheus/prometheus

For our test case, we need only 1 consumer, to make the request, one payment system, to approve the payment and generate the token for the utilization, and N cameras, that the consumer wants to use.

Using a computer CPU Intel(R) Core(TM) i7-3632QM CPU @ 2.20GHz and 16GB of RAM, the theoretical limit of our test is 60 machines running (58 cameras + 1 consumer + 1 miner) with the limiting factor being the CPU.

5 THE SWARM COMPUTING PARADIGM AND THE SWARMOS FRAMEWORK

Swarm is a distributed network of cooperative devices. The swarm computing concept expands the IoT to an organic network whereby the characteristics, communication, and topology of the agents can be adapted dynamically. It particularly considers IoT devices as agents that act independently.

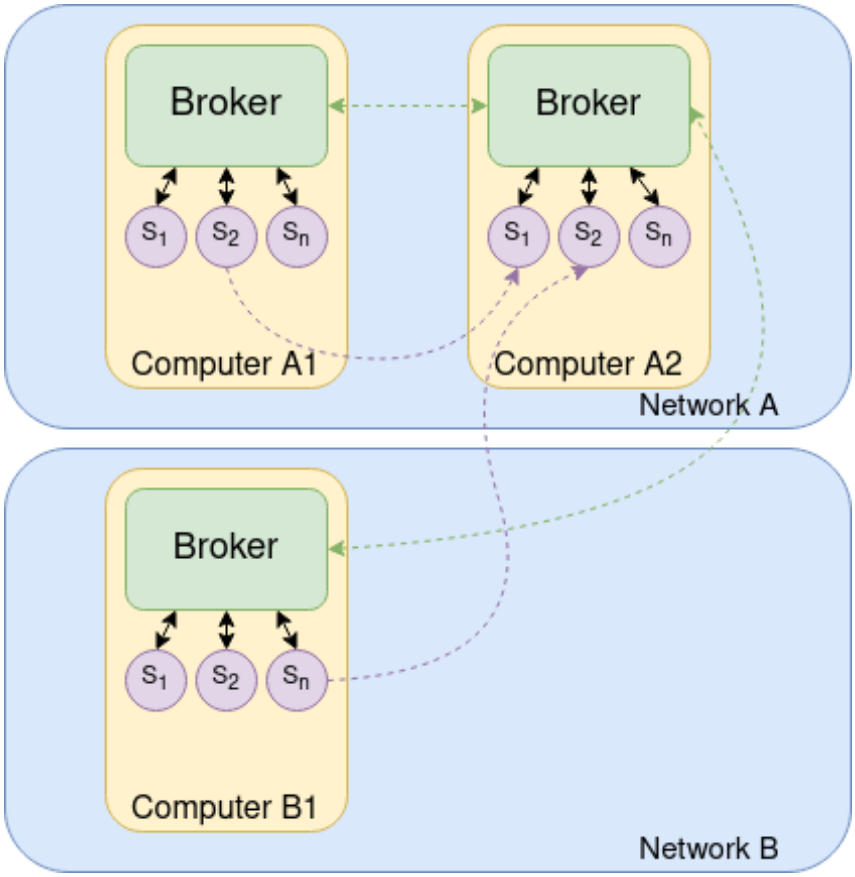
A swarm system is a decentralized, self-organizing network of entities that work together to achieve a common goal. In a swarm system, individual entities, often referred to as agents or nodes, interact locally with their neighbors based on simple rules, and the collective behavior emerges from these interactions.

SwarmOS is a framework composed of services needed for creating swarm systems. SwarmOS enables the deployment, scaling, and management of decentralized applications and services, intending to make it easier to build, deploy, and operate large-scale, decentralized systems. The operating system provides a set of tools and services to manage and coordinate the interactions between nodes in the swarm, as well as an API for developing decentralized applications that can run on top of the SwarmOS. The goal of the SwarmOS is to provide a flexible and scalable platform for building decentralized systems that can handle the growing demands of the IoT devices and applications. The architecture is composed of many devices each one with one Swarm Broker and one or more microservices, also called Swarm Agents.

A Swarm Agent is any autonomous entity that drives its activity towards achieving goals, they are usually service consumers or service producers (COSTA et al., 2015a). Brokers provide common resources to other agents, such as discovery, mediation, security, and contract establishment (COSTA et al., 2015a). In figure 11 we can see a generic swarm architecture.

The SwarmOS structure is developed using REST communication (COSTA et al., 2015a). The applications developed must follow a protocol compatible with the broker, exposing REST endpoints the broker uses to mediate. An example of communication

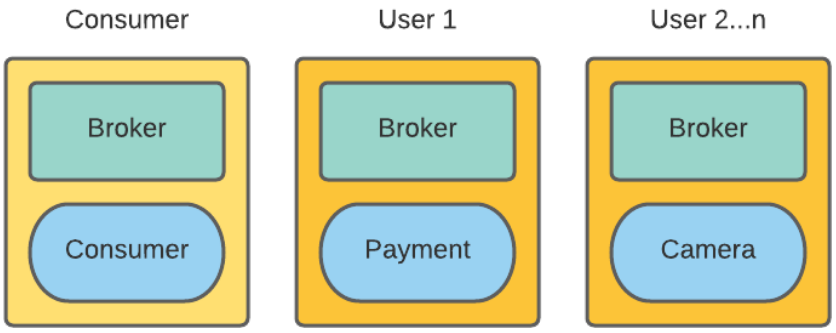
Figure 10: Examples of 3 types of communication that exist in the structure of SwarmOS. In green (dotted), communication between brokers; in black (solid) communication between broker and services; in purple (dashed), communication between services



in a common architecture can be seen in Figure 10. A single broker can have n services. A service starts by registering with the broker's registration system, the broker uses its tools to search for other brokers. When a service or broker needs to use an external functionality, it creates a query and executes it by starting the contract in known networks or brokers. A diagram of how the broker understands the network structure is shown in Figure 11.

In order to facilitate the creation of new applications and communication with the broker, the SwarmOS project has incorporated a library, specifically developed in the Python programming language. This library facilitates accessibility to the Swarm network by providing default functions that implement the SwarmOS communication protocol. These functions act as conduits, facilitating interaction between various entities within the system, including applications and brokers, as well as inter-application communication.

Figure 11: Diagram of the structure the broker discovered.



5.1 SwarmOS Transaction Model

The SwarmOS transaction model aims to facilitate the interaction between service providers and consumers and ensure efficient and reliable service provision in a swarm network. A key element in the development of a cooperative network within the Swarm centers on the economic domain. Swarm Economy aims to ensure fair conditions that encourage the sharing of resources among all participants. The economic facet becomes increasingly complex in networks with distributed governance, as maintaining fairness and transparency becomes a more intricate task. Such challenges stimulate the adoption of advanced technologies capable of providing a trustworthy and immutable public registry (BIASE et al., 2018).

In any system where monetary transactions or incentives are involved, there is an increased likelihood of encountering malicious nodes. These malicious entities can be driven by the prospect of achieving unfair advantages or illicit gains. They employ various strategies such as exploiting loopholes in the system, spreading disinformation, or manipulating services.

The presence of malfunctioning agents is inevitable due to various factors such as software bugs, hardware failures, or network disruptions. These malfunctioning agents can adversely affect the network's performance, causing delays and hindering overall scalability. To ensure the network's scalability and smooth operation, it becomes essential to expel these malfunctioning agents promptly. Removing malfunctioning agents enables the network to allocate its resources more effectively, optimize communication pathways, and foster a more efficient ecosystem for genuine and reliable participants to interact, ultimately enhancing the network's scalability and overall performance.

The difficulty of finding a solution without relying on a central server to ban nodes makes research in this area an ongoing and prominent topic. Addressing bad actors and malicious nodes in a decentralized manner poses challenges, as it requires innovative and distributed techniques to maintain network integrity and security. As technology evolves, researchers continue to explore novel approaches and decentralized mechanisms to effectively deal with bad actors in P2P networks, making it a persistent and relevant area of investigation.

The SwarmOS implements a model for transactions based on nine steps: Registration, Service Request, Discovery, Select Optimum Service Provider, Negotiation, Payment, SLA Establishment, Feedback and Reputation Points Attribution, and Use (BIASE et al., 2018).

In short, the provider registers its services by sending a description file to the consumer's broker providing information such as the type of service, its capabilities, and its price (1. Registration). Because SwarmOS uses a price-reputation model, the price is not flat, it is calculated based on the level of trust of the provider in the consumer, this price calculation rewards good behavior and penalizes bad behavior. When a consumer wants to use a service, it sends a query to their consumer's broker with some parameters, specifying parameters like the type of service, price limit, and requested time window in which the service will be used (2. Service Request). The consumer's broker then takes this query and receives a list of available service providers that fulfill the query parameters (Explained in more detail in section 5.2) (3. Discovery). The consumer's broker then selects the provider with the lowest price (4. Select Optimum Service Provider). During the negotiation, the consumer's broker sends to the provider's broker a transaction with a Service Level Agreement (SLA), with price, duration, consumerID, etc. (5. Negotiation). If the SLA is accepted, the consumer's broker will conclude the trade with the payment done using a virtual currency based on blockchain [1] (6. Payment). Now that the payment is done and confirmed, the provider's Broker will issue permission to grant access to the consumer to the provider (7. SLA Establishment). then in the final step, the consumer can use the permission granted to use and provide feedback on the service on a blockchain (8. Feedback and 9. Use).

In the study (BIASE et al., 2018), the author introduces a formula for calculating reputation feedback as follows:

$$Points_i = \alpha F_i + (1 - \alpha) F_b$$

where $Fi \in [0, 5]$ and represents the feedback from another agent, which is based on

[1] https://lhartikk.github.io/

the quality of information provided, $Fb \in [0, 5]$ and symbolizes the broker's feedback, which is based on the broker's role in the transaction process (such as negotiation and payment). Lastly $\alpha \in [0, 1]$ where $\alpha = Rep_i/10$. However, the consumer's reputation, 'Rep_i', is not yet implemented in SwarmOS version 0.2.3. Without this component, the calculation for 'α' becomes infeasible. Consequently, a default value of $\alpha = 0.8$ has been used for the time being.

5.2 SwarmOS Discovery Model

Before the present development, SwarmOS was limited to functioning solely within local networks. Its modus operandi was straightforward - it created a query which was transmitted via multicast to all brokers within the same network. In response, it received a list of providers that satisfies the given query parameters, streamlining the selection process.

One interesting feature of the pre-existing SwarmOS was the concept of query redirection. Each query carried with it an integer value, representing the number of redirections for that particular query. For instance, the original requester would send a query with the field redirection number as zero, the brokers that initially received this query would increase the number by 1 and forward the query to the multicast. Essentially, they acted as secondary requesters, forwarding the same query to their network. The responses were then compiled and returned to the requester.

This method, though somewhat rudimentary, allowed for the broadening of search parameters and potential service providers. However, it was still constricted by the limitation of functioning within the local network. As we move forward, we will discuss how the revised SwarmOS overcomes these restrictions and enhances overall network functionality and service discovery.

5.3 SmartABAC

SmartABAC (FEDRECHESKI et al., 2021) is an innovative model for access control in the IoT that is designed to run directly on IoT devices, aligning with the trend toward IoT decentralization.

To simplify and enhance the policy parsing process, SmartABAC utilizes the enumeration-based access policies (EAP) (BISWAS; SANDHU; KRISHNAN, 2016). To address the inherent

limitations in enumerated models, it incorporates typed and hierarchical attributes as a compensatory measure.

Enumeration-based access policies in ABAC involve defining a finite set of attributes and their possible values. These attributes can represent various characteristics of users, devices, resources, or environmental factors relevant to access control decisions. The enumerated values associated with each attribute determine the specific permissions or restrictions granted to entities.

For example, in an IoT system, attributes like "device type", "location", or "user role" can be enumerated with specific values such as "temperature sensor", "living room", or "admin". Access policies are then defined based on the combinations of attribute values, specifying which entities are authorized to access specific resources or perform certain actions.

Enumeration-based access policies provide a simplified and efficient approach to access control management. By predefining a finite set of attribute values, policy administration becomes more manageable, as administrators do not need to define complex rules or conditions for each individual access decision. However, it is important to carefully design and maintain the enumeration values to ensure they accurately represent the relevant attributes and access requirements of the system.

One limitation of enumeration-based policies is their lack of flexibility and expressiveness compared to more complex policy languages or models. They struggle to accommodate dynamic or fine-grained access control requirements that require more nuanced decision-making. But because ABAC models based on logic statements require highly specialized parsers, they can be NP-complete to audit (BISWAS; SANDHU; KRISHNAN, 2016).

5.4 Improvements in the SwarmOS Implementation

5.4.1 Improvements in SwarmOS connection system

The SwarmOS version 0.2.3 uses a discovery system based on two in-memory lists, one for internal agents and another for remote agents. Every broker can also work as a registry for other users' services. The discovery service can be divided into four functions; (1) Internal agent in-memory list search; (2) Remote agent in-memory list search; (3) Registry request search; and (4) Multicast search.

The current strategy employed by SwarmOS appears effective within local networks, but it faces challenges when peers attempt to find each other across distinct networks. This limitation arises from NATs that have the ability to assign private IP addresses to devices within a local network, while presenting a single public IP address. This results in communication barriers, as external devices cannot directly initiate connections to devices within the local network without a workaround.

Due to this constraint, SwarmOS version 0.2.3 possesses a latent limitation concerning global connectivity, being primarily confined to local networks. To address this limitation, two techniques are proposed to enable global connectivity for SwarmOS, leveraging the Internet's capabilities: the UPnP protocol and support for IPv6. These additions aim to enhance the network's reach and ensure seamless communication between peers across various networks.

5.4.1.1 Adding Universal Plug and Play (UPnP) Support

In the section 2.2 we explained the difficulties of connecting devices globally because the Internet was designed with a client-server relationship in mind, also the existence of Network Address Translation (NAT) devices, NATs are used to map internal private IP addresses to a public IP address. This is done to conserve the limited number of public IP addresses and to provide a layer of security for internal devices. However, NATs can cause difficulties when it comes to establishing P2P connections because they often prevent incoming connections from being established. This can lead to devices being unable to connect to each other, making it difficult to form a global P2P network.

One solution that this work brings to the table is to add UPnP support improving its connectivity by making it easier for devices to communicate with each other even when behind Network Address Translation (NAT) devices such as routers.

UPnP includes mechanisms to enable devices behind NAT routers to open temporary ports for communication with external devices, facilitating P2P connections even across different networks. This mechanism helps devices bypass NATs by allowing them to automatically configure their NAT devices to forward incoming P2P traffic to the appropriate internal IP address. This improves the overall connectivity of the P2P network, making it easier for devices to connect with each other and exchange information.

Due to the inherent complexity in configuring NATs, the inclusion of UPnP support proved to be beneficial:

- Improved Network Connectivity: UPnP allows devices to automatically configure network settings, making it easier for devices to connect to the Internet and communicate with each other. This can greatly improve network connectivity and make it easier for devices to work together.
- Dynamic Configuration: UPnP allows devices to dynamically configure network settings, such as port forwarding, which can be useful for applications that need to communicate through firewalls and NATs (Network Address Translation).
- Improved User Experience: UPnP simplifies the process of setting up and configuring devices, making it easier for users to get their devices up and running quickly. This can improve the overall user experience and make it easier for users to use and enjoy their devices.

Overall, implementing UPnP can help to improve the functionality and reliability of a network, making it easier for devices to communicate and work together effectively. The implementation is described on section 5.4.1.3.

5.4.1.2 Adding IPv6 Support

The creation of IPv6 has eliminated the necessity for a unified security architecture, which was once required in IPv4 due to addressing constraints imposed by NAT. As a result, manufacturers now have the freedom to choose between two design models. They can opt for an open model that enables devices to be directly reachable end-to-end, or they can stick to a familiar closed model in which the home gateway acts as a middleware device. This shift presents a complex environment with various nuances like different filtering behaviors, diverse access control policies, and specific IPv6 requirements, resulting in an ambiguous setting. (OLSON; WAMPLER; KELLER, 2023)

But NAT will not disappear, the default NAT architecture undoubtedly offered a plethora of privacy and security benefits for non-technical users, establishing a nearly universal operational baseline. This made it easier for users with limited technical expertise to enjoy enhanced online protection and privacy.

For developers, the potential impact of these changes is unclear. Like NAT, the technology will be developed together with the new requirements that new features will bring. Swarm and P2P networks can benefit a lot from an open model design, where any node can communicate with any node, but this will depend on how the Internet

evolves from now on. The absence of standards makes it more challenging to determine the specific scenarios in which IPv6 and SwarmOS will seamlessly work together.

In this work, I developed support for IPv6, enabling SwarmOS to leverage the advantages of this new protocol. With IPv6, the addressing space is expanded, and every device can have a unique IP. By implementing IPv6 support, SwarmOS can now take advantage of these features and provide a more robust and secure infrastructure for IoT devices.

5.4.1.3 Implementation UPnP

As explained before, UPnP is a type of port-forwarding technology. The application sends a message to the NAT device asking for an external IP, the NAT then creates a rule to forward packets that come from this IP:port to the application.

For this work we will leverage the mechanisms to enable brokers behind NATs to open temporary ports for communication with external devices.

The SwarmOS is written in Elixir, a functional programming language built on the Erlang Virtual Machine (BEAM). This design enables seamless interconnection with Erlang, allowing SwarmOS to take advantage of the vast ecosystem and robust features offered by the Erlang platform. The library utilized was erlang-nat that provides efficient NAT handling facilities.

While using erlang-nat a minor issue arose due to its use of httpc, an Erlang module that provides the API for HTTP/1.1. However, the SwarmOS elixir code, with its more contemporary approach, relies on hackney for HTTP functionalities. As part of my contributions to the project, I updated erlang-nat to use hackney, leading to the creation of a forked version available at https://github.com/WilliamTakeshi/erlang-nat.

Subsequently, an additional module was integrated into SwarmOS. During the initialization of SwarmOS, the broker checks if the NAT router implements the UPnP protocol. If the UPnP protocol is supported, the broker creates a new port map effectively bridging the external IP address and port to the internal one, enabling exposure to the external network.

The Python implementation was also executed using the swarm_lib_python. A new module was developed within the swarm_lib_python to facilitate the exposure of the agent to the external network during the initialization phase. For this purpose, the upnpy library was utilized for the communication with the UPnP protocol.

Upon integrating both modules, no additional effort is required for the developers.

They can seamlessly utilize UPnP without any impediment, simply by enabling a single flag. This streamlined process grants developers the freedom to harness the benefits of UPnP effortlessly within their applications.

If the flag is enabled, the router will establish a mapping for the Swarm Agents and the Broker, and subsequently, the service description will be updated with a new entry in the service field.

Listing 5.1: UPnP example

```
{
        "serviceEndpoint": "http://100.68.138.64:5018/",
        "id": "#upnp",
        "type": "swarmService"
}
```

Also, any other swarm agent will have the possibility of trying a connection with the UPnP port.

5.4.1.4 Implementation IPv6

In order to integrate IPv6 support into SwarmOS, a novel module was incorporated within the SwarmOS broker. This was coupled with a Python library, previously discussed, which offers us a straightforward mechanism to establish a service connected with the broker. This module and library together identify if an IPv6 address is accessible and then add this information to the service list.

The inherent functionality of SwarmOS ensures that the system will automatically seek out this IP address and assess its connectivity. Should it receive a positive response, the IPv6 address will be designated as the endpoint for all subsequent communication.

Listing 5.2: IPv6 example

```
{
        "serviceEndpoint": "http://[1ffe:0:a8:85a3::ac1f]:501
           6/",
        "id": "#ipv6",
        "type": "swarmService"
}
```

5.4.2 Improvements in SwarmOS's Discovery System

This work proposes a discovery system for SwarmOS, premised on the tracker model frequently employed within BitTorrent P2P networks. The primary objective of this system is to enhance the efficiency and scalability of device discovery within a decentralized IoT framework. The work will dive into an evaluation of this discovery system's performance via testing and experimentation. Additionally, a series of minor improvements and bug fixes have been proposed. By juxtaposing the performance of our system against existing discovery mechanisms, our goal is to underscore its proficiency in facilitating seamless and reliable peer discovery within the context of expansive P2P networks.

In a P2P torrent network, a tracker serves as a crucial intermediary that facilitates the connection between peers. When a peer joins a torrent, it communicates with the tracker, which provides a list of other peers currently downloading or seeding the same torrent file. This system ensures efficient data exchange by continually updating and sharing peer lists, thereby maintaining a robust and effective network for file sharing.

Because of the dynamism of the Swarm network, with new devices, and applications joining and leaving the network all the time, users may not be aware of all resources and services provided by the network, so resource discovery is a fundamental tool to facilitate and scale interactions among swarm devices.

To have a scalable technology, one important dimension is the degree of distribution. This limitation can define the impact networks can have on massively distributed systems (CCORI et al., 2016). So it is important to have the technology to distribute queries.

A common approach to create components reusable and interoperable is service-oriented architecture. The discovery of things, their resources, proprieties, and capabilities is a requirement to enable the transactions on the network.

The Swarm network employs the microservice approach, a modular design pattern that breaks down applications into smaller, independent components. Within this architecture, each microservice can make resources available, enabling consumers to utilize, combine, and adapt these resources to create innovative solutions tailored to their specific problems or needs.

We will now delve in a detailed explanation of how the SwarmOS discovery system proposed and implemented works. This examination will provide insights into the underlying mechanisms that will be useful to understand the test being made.

Table 5: Comparison of Centralized, Pure, and Super-Node resource management models. Adapted from (KHATIBI; SHARIFI, 2021)

	Search Performance	Scalability	Resilience to Single-Point-of-Failure
Centralized Model	High	Low	Low
Pure Model	Low	High	High
Super-Node Model	Medium	Medium	Medium

Resource discovery in P2P networks can be classified into three main models. The first is the centralized model, in which all information is stored on a single server or a cluster of servers. The second is the pure model, where every node operates as both a server and a client. Lastly, there's the super-node model, wherein a select group of nodes, referred to as super-nodes, oversee and manage a smaller subset of nodes. (KHATIBI; SHARIFI, 2021).

In our case we choose a pure model, where every broker works as a registry, that means they have the service that maintains a list of all available services within a particular network or system. The registry works with two in-memory lists, one for internal agents and another for remote agents. This distinction is necessary due to the variance in the service descriptions between local and remote agents. Specifically, local agents possess a more comprehensive set of information in their fields, necessitating a separate list to account for these additional details like private credentials that are not needed on remote agents.

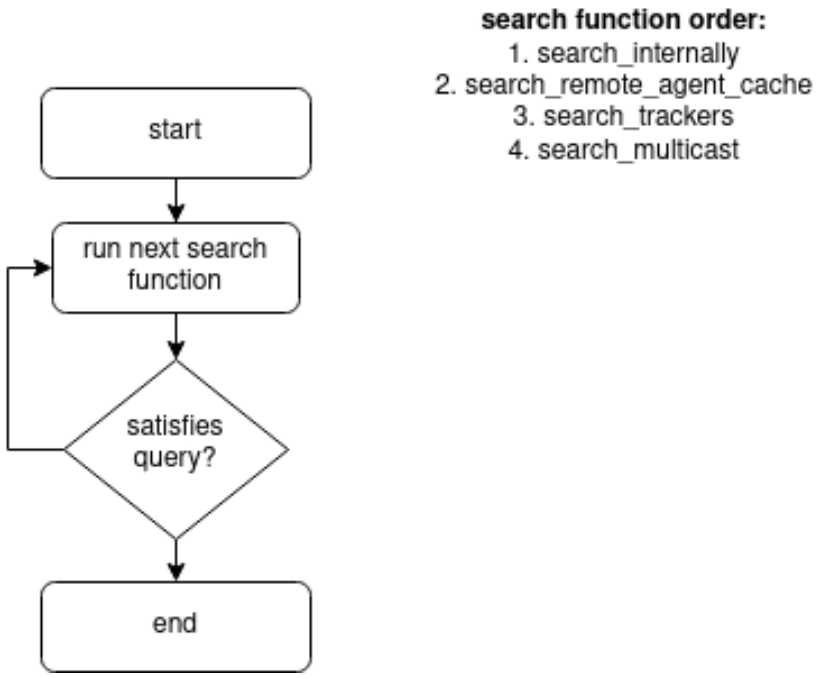
The discovery service works using four different functions:

1. Internal agent in-memory list search
2. Remote agent in-memory list search
3. Registry request search
4. Multicast search

If the query is fulfilled by any given function, the broker ceases the search and does not proceed to the next function. For instance, if the query successfully retrieves all requested services from the first function ("Internal agent in-memory list search"), it will prevent the execution of all subsequent functions. The behavior is represented on Figure 12

The first two functions ("Internal agent in-memory list search" and "Remote agent in-memory list search") search matches inside in-memory lists. The "Internal agent in-

Figure 12: SwarmBroker discovery fluxogram



memory list search" are agents registered on their broker (Fig 13), the "Remote agent in-memory list search" are remote agents, added to the in-memory list by results of old queries or other fellow brokers registering their agents as remote agents (Fig 14).

The initial pair of functions, "Internal Agent In-Memory List Search" and "Remote Agent In-Memory List Search", are responsible for seeking matches within the respective in-memory lists. The "Internal Agent In-Memory List Search" refers to agents that are registered within their own broker (Fig. 13). Conversely, the "Remote Agent In-Memory List Search" relates to remote agents, which are appended to the in-memory list either as a result of previous queries or through fellow brokers registering their own agents as remote agents (see Fig 14 for a detailed representation).

The last two functions ("Registry request search" and "Multicast search") send queries for known registries or broadcast on their multicast network. If the registry or the multicast match the query, they return a list of results that the broker who did the initial query can assemble and decide the ones they wanna use. The "Registry request search" is displayed on Fig 15.

The last pair of functions, namely "Registry Request Search" and "Multicast Search", execute queries either to known registries or broadcast them on their multicast network. When the registry or multicast matches the query, they return a list of results. This list is then received by the querying broker, who is able to assemble these results and selectively

Figure 13: Block diagram of searching on the internal cache

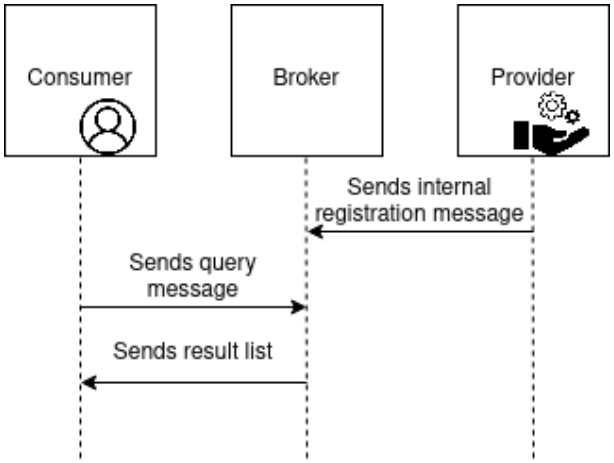
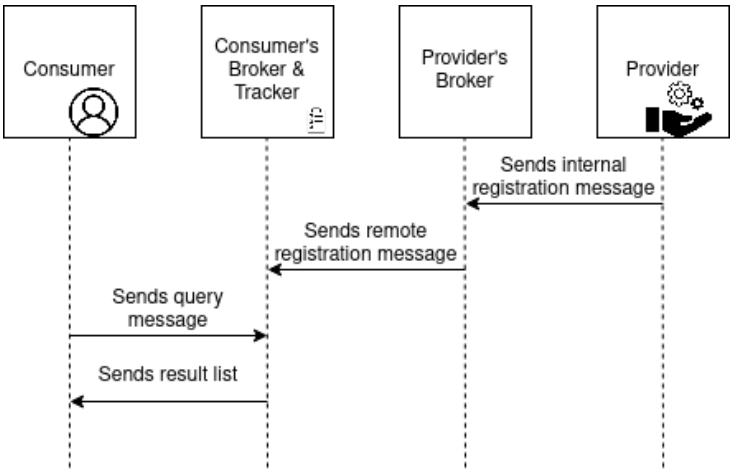


Figure 14: Block diagram of searching on the remote agents' cache



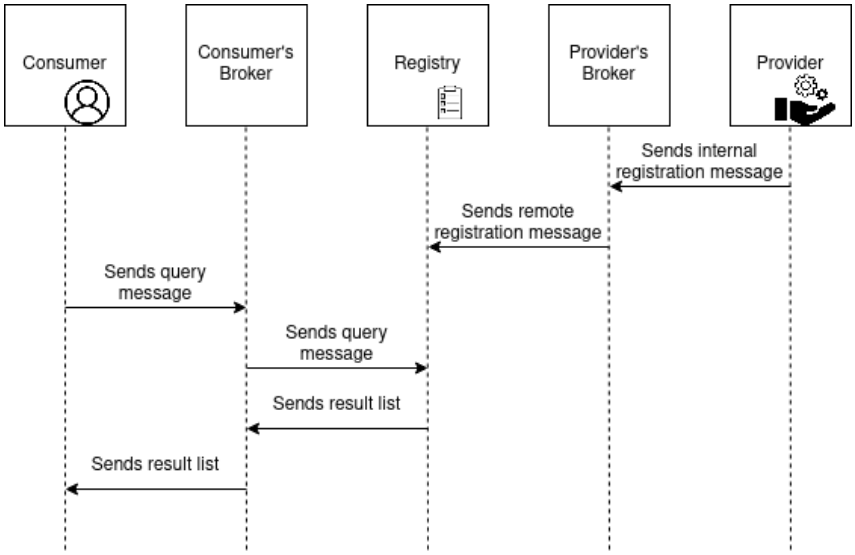
utilize the services that best align with their requirements. Also, if the querying broker wants, it can save the result in the remote in-memory list of agents. A visual representation of the "Registry Request Search" function can be found in Fig 15.

5.4.3 Improvements in Swarm Manager

Swarm manager is a command line interface (CLI) software tool to help configure, create, update, and delete agents on a swarm network. It provides a convenient interface for administrators and developers to manage and interact with their swarm agents, without the need for manual, low-level operations. By using the swarm manager, users can easily create and edit their IoT devices within the swarm network, ensuring reliable and efficient communication and coordination among agents and decreasing the chance of bugs that can appear with manual editing of the configuration files.

With this tool, managing a swarm system becomes more streamlined, making it easier

Figure 15: Block diagram of querying the registry



for users to scale their IoT solutions and achieve their goals.

Because of our tests we had to create hundreds of different devices, each one having different behaviors, the use of swarm manager was vital to streamline the process and ensure that each device was configured correctly. Without the use of a swarm manager, setting up and managing such a large number of devices would have been extremely time-consuming and prone to errors. The use of swarm manager allowed us to automate many of the tasks involved in configuring and managing the devices, making our testing process more efficient and less prone to errors.

In this work, new features were added to the Swarm Manager, for example, routines to update IPs. On swarm networks, it is not uncommon to have devices changing locations all the time, and manual updates can become time-consuming and difficult to manage. Having devices broadcasting the wrong ID increases the time to find a suitable application on the query system. Also creating functions that take long and are prone to error tasks and automating, for example, the problem of issuing credentials. Since on the SwarmOS network, we have to take the value issued by one broker and give it to another application, this can be really hard, especially for thousands of different devices and applications.

5.4.4 Caching

Cache is a type of memory storage in computing that temporarily holds frequently accessed data, enabling quick data retrieval on demand. In the context of SwarmOS, we utilize caching as a strategy to reduce the overall number of network exploration requests.

This application of caching aims to enhance system performance by minimizing redundant network queries and thereby decreasing latency. However, it's important to manage caches correctly as outdated or stale cache data can lead to inconsistencies.

Rather than sending a fresh request every time a user searches for a specific service, a cached response can be provided if available, resulting in reduced network load and discovery time.

The SwarmOS discovery system borrows the idea of connectivity checks from ICE protocol (Interactive Connectivity Establishment)(ROSENBERG et al., 2012). ICE works by gathering all available IP addresses and ports. The gathered candidates are then tested by trying to establish a connection, and the most efficient, successful route is selected for the connection.

In similar fashion, SwarmOS devices share their respective lists of candidate IP addresses and ports with each other. Subsequently, they begin performing connectivity checks by mutually exchanging data packets via the candidate addresses belonging to the remote device. The first route that successfully facilitates data exchange is selected for establishing the connection and the connection details are stored, or 'cached', for future use.

For instance, when a service description contains multiple endpoints for various types of communication (local, LAN, UPnP, IPv6, etc.), caching the endpoint that previously functioned successfully can significantly enhance response speed.

This results in a faster, more responsive system and improved overall user experience. Additionally, caching can also help to reduce the cost of system operations by reducing the number of resources required to access the data.

5.4.5 Privacy

Writing internal tools to automate manual tasks can greatly benefit a system in terms of efficiency and productivity. By automating repetitive tasks, developers can free up their time to focus on more important and value-added activities. Additionally, internal tools can reduce the risk of human error, especially access control/security where many steps can be required and mistakes can cost internal data being leaked.

One important contribution of this dissertation was to streamline development processes and reduce manual errors by developing internal tools. With the help of these tools, developers can focus on more critical tasks, such as design and testing, while automated

tools take care of the routine tasks. This leads to a reduction in the overall development time and improves the quality of the final product. Furthermore, having these internal tools in place also makes it easier to invite new developers and maintain consistency across the network. Overall, the development of internal tools is a crucial step toward creating a more efficient and effective development process.

6 RESULTS AND EVALUATION

In this section, we evaluate the SwarmOS across four key dimensions: connection, discovery, fairness and privacy. Through various use cases, the system's ability to scale is tested, and the performance of its newly added features is examined.

6.1 Connection

6.1.1 Testing Device Interactions in Varied Network Environments

The experimental approach involved an assortment of network environments to test the communication capabilities of multiple physical devices dispersed across different networks. The aim was to determine if devices, pre-equipped with knowledge of each other's service descriptions, could effectively communicate across diverse network scenarios, employing a variety of communication strategies.

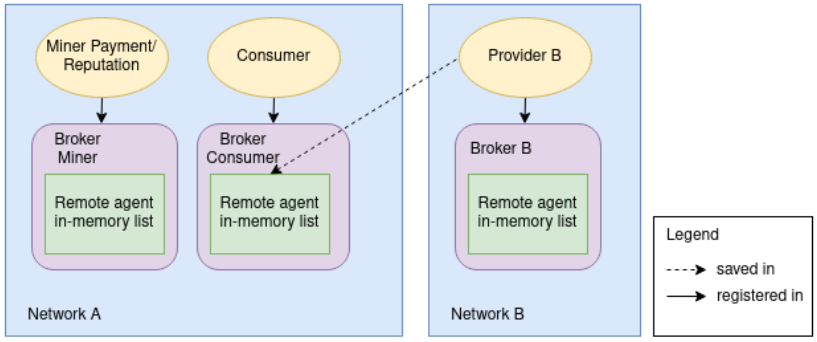
The networks employed for the experiment varied from direct connections to public internet networks to controlled home networks located behind NATs. The devices were pre-programmed to know each other before being distributed. The communication methodologies encompassed a wide spectrum, starting with localhost checking, moving on to intra-network communication, and finally resorting to using public IPs through UPnP or IPv6 when necessary.

Our experiments furnished valuable insights into the practical capabilities and limitations of device communication across various network contexts. These findings play a crucial role in informing future network configuration strategies, enhancing device com-

Table 6: Results of the Device Interaction test

Connection Type	**Provider**	**Successful**
UPnP	Swarm:Camera	✓
IPv6	Swarm:Camera	✓

Figure 16: Structure of a Device Interaction test



munication algorithms, and overall network performance across a diverse range of devices and network conditions.

6.1.2 Concurrent Connection Capacity Test

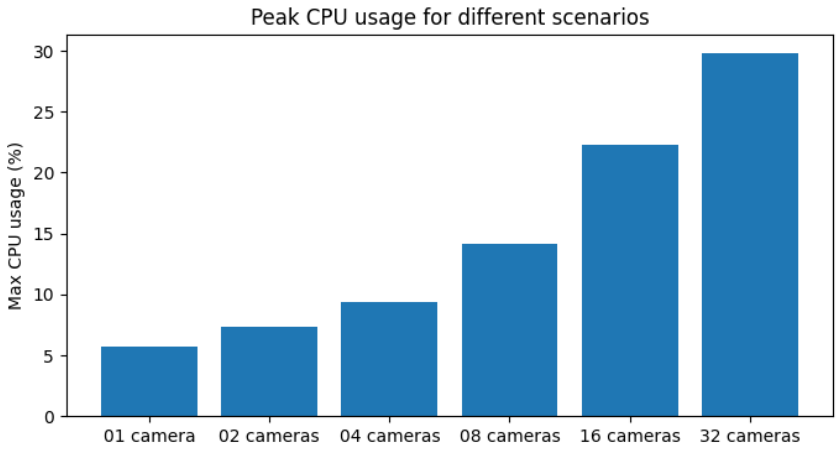
The Concurrent Connection Capacity Test serves a dual purpose. The principal focus is to ascertain the code's ability to manage and maintain numerous simultaneous connections seamlessly. Secondly, it evaluates the scalability of our testbed to determine its capacity limits.

Initially, our setup comprises a single consumer, one provider represented as a camera, a blockchain dedicated to payment, and another for reputation management. As we progress through different scenarios, we double the count of providers in each step. Starting with just one, we continue this increment until we reach a total of 32 cameras, allowing us to observe the system's behavior with escalating numbers of providers.

The results were derived using a testbed that employed cadvisor in conjunction with prometheus. These tools collaborated to monitor and gather the essential metrics of our system. Specifically, we extracted values for three key parameters: `container_cpu_user_seconds_total`, which tracks the total CPU time used; `container_network_receive_bytes_total`, indicating the total bytes received over the network by the container; and `container_memory_usage_bytes`, which measures the memory consumed by the container. This approach ensured to capture a holistic view of the system's performance under varying conditions.

During each test, 10 transactions are executed in which the system actively searches for every available camera within the network. Once these cameras are identified, the system proceeds to use them. This process ensures a thorough assessment of the network's

Figure 17: Peak percent CPU usage of a single core in scenarios varying from 1 to 32 cameras



responsiveness and the efficiency of the hiring mechanism across multiple transactions.

To determine the code's capability to handle multiple simultaneous connections smoothly, the test results were positive. The consumer successfully connected with all 32 providers without any issues.

6.1.3 Results

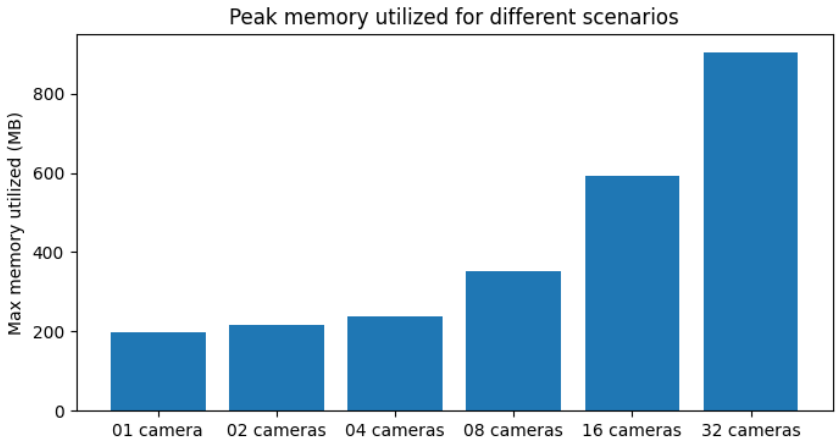
Firstly we will start with `container_cpu_user_seconds_total` .The metric represents the total amount of CPU time, in seconds, that processes in the container have spent in "user" mode since the container started. When a process runs in "user" mode, it is executing user-written code, as opposed to "system" mode, where the CPU is executing system/kernel code.

Since we are focusing on a scalability analysis of the SwarmOS, we choose specifically the user mode to not take in account the system time that docker can add in the CPU usage.

The CPU usage for the consumer increases proportionally with the number of cameras added to the network. This trend is logical, as the consumer interacts with more brokers, establishes additional connections, and performs related tasks. Using the `container_cpu_user_seconds_total` we can derive the percent of a core used for the experiment.

By analyzing the `container_cpu_user_seconds_total`, we can determine the percentage of a single core utilized during the experiment. Figure 17 depicts the peak CPU usage for a single core across different test setups. It's important to note that, given the test machine has 8 cores, the maximum attainable value is 800%.

Figure 18: Peak memory usage in Mb in scenarios varying from 1 to 32 cameras



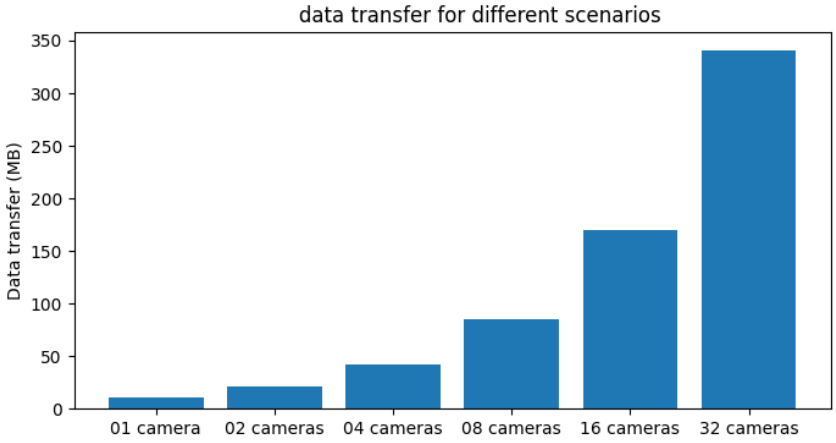
Turning our attention to the `container_memory_usage_bytes` metric, it measures the total memory presently consumed by a container, encompassing all memory regardless of its last access time. This metric is instrumental in gauging the real-time memory footprint of containerized applications or services. Given the dynamic nature of memory allocation (with memory being allocated and freed as necessary), we captured the peak value, which typically occurs during a transaction.

By examining the `container_memory_usage_bytes`, we can pinpoint the highest memory consumption in MB. Figure 18 illustrates the peak memory used by the consumer when retrieving camera images across various provider configurations. It's noteworthy to mention that the test machine was equipped with up to 16MB of memory, ensuring that the testbed was never resource-deprived at any point.

Lastly, let's discuss the `container_network_receive_bytes_total` metric. This metric tallies the cumulative number of bytes a container receives through its network interfaces. Being an accumulative counter, its value continually increases unless the system undergoes a restart or the counter is reset. Monitoring this particular metric provides insights into the inbound network traffic volume of a specific containerized application or service.

By examining the `container_network_receive_bytes_total`, we can gauge the network traffic. It's important to note that each provider generates an image of a fixed size, specifically 8MB.

Figure 19: Data transfer for different scenarios varying from 1 to 32 cameras



6.1.4 Scalability Analysis

The network's connection system demonstrates considerable strengths in addressing diverse challenges, as unveiled during a comprehensive scalability analysis.

Primarily, the brokers showcase an automatic capability to detect accessible protocols and dynamically incorporate them into their service descriptions, enabling seamless connections with other brokers. This adaptive behavior streamlines the user or developer's tasks by providing a self-configuring environment.

Moreover, the concurrent connection test highlights that resource utilization scales linearly with the number of connections. The system exhibits efficient handling of numerous concurrent connections, attributed to its REST architecture. This design choice facilitates scalability by efficiently managing the communication demands in a manner that aligns with the growing number of connections. Overall, the connection system not only showcases adaptability in addressing diverse protocols but also underscores its scalability through efficient resource utilization in handling concurrent connections.

In the work Kash et al. (2012), shed light on the seeding and leeching behavior in the context of BitTorrent. Despite a small percentage of users engaging in a high number of concurrent seed/leech, the median leecher typically participates in just 1 torrent, and the median seeder is active in 4 torrents. This data underscores that, in practical terms, many users do not engage in numerous concurrent connections. Drawing parallels from this BitTorrent example, it suggests that for certain applications or systems, a substantial number of concurrent connections might not be a common or necessary scenario. The study highlights the importance of understanding the specific requirements and usage patterns within a network to optimize resources effectively.

6.2 Discovery

6.2.1 Caching Performance Test

As delineated in section 5.2, the SwarmOS operates by sequentially attempting every possible endpoint. Upon receiving a valid response from an endpoint, it's memoized for prioritized use in subsequent transactions. Prior to the implementation of caching, this method was performed every time a new transaction was required. The process was inevitably time-consuming and somewhat inefficient.

The objective of this test is to monitor the system in action and address potential issues arising from the newly introduced feature.

6.2.1.1 Results

With the implementation of caching, the system became significantly more efficient. Now, based on the information from the last successful transaction, the SwarmOS already knows which endpoint works, eliminating the need to sequentially test every endpoint each time.

Below we represented an example service list and the time taken to resolve the endpoint in relation to the position of the first functional endpoint in the list. The position index helps us understand how far into the list we had to traverse before finding a functional endpoint, and thus, the time taken for endpoint resolution.

This comparative analysis clearly demonstrates the improved efficiency and time-saving advantage offered by the caching mechanism. It reinforces the utility of caching in facilitating faster and more effective transactions in SwarmOS.

This is a sample service endpoint list, a field that provides potential consumers with a range of endpoints they can utilize to request a service. Because the consumer can't ascertain whether they reside on the same localhost or intranet, they must attempt each endpoint in sequence until they receive a successful response.

Listing 6.1: Caching example

```
service: [
  %BrokerCore.Registry.Service{
    id: "#localhost",
    serviceEndpoint: "http://localhost:5151/",
    type: "swarmService"
```

```
  },

  %BrokerCore.Registry.Service{
    id: "#lan",
    serviceEndpoint: "http://192.168.100.104:5151/",
    type: "swarmService"
  },

  %BrokerCore.Registry.Service{
    id: "#upnp",
    serviceEndpoint: "http://100.68.138.126:5151/",
    type: "swarmService"
  },
   %BrokerCore.Registry.Service{
    id: "#ipv6",
    serviceEndpoint: "http://[1ffe:0:a8:85a3::ac1f]:5151/",
    type: "swarmService"
  }
]
```

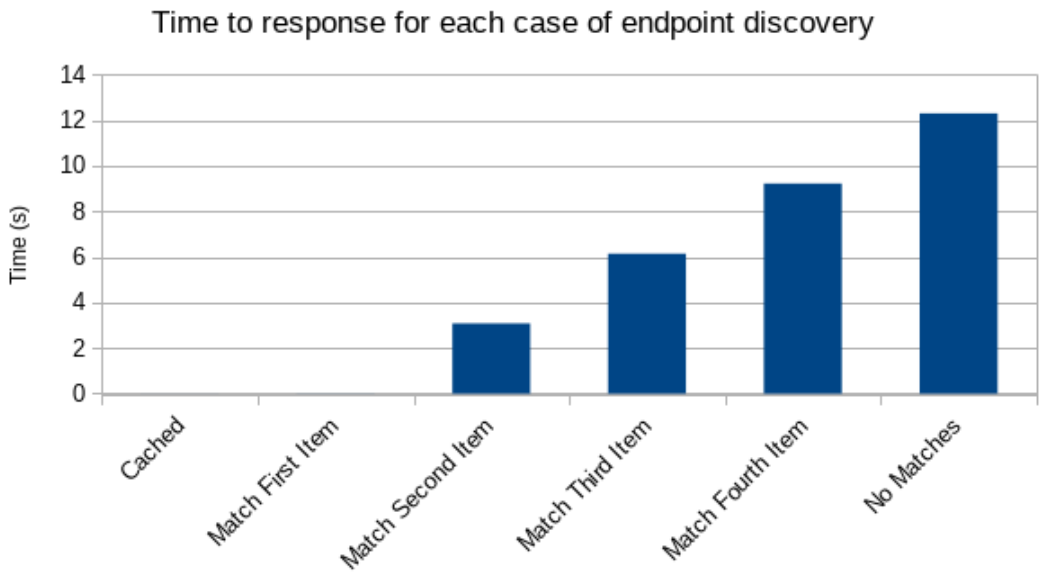
After a transaction is completed, the subsequent transaction utilizes the cached endpoint, significantly enhancing the transaction speed. For instance, when a consumer and a provider are on different networks, neither the localhost nor the intranet address serve as valid endpoints. The first viable endpoint is the UPnP, but pinpointing this endpoint by testing and discarding the initial two takes slightly more than 6 seconds. By caching, the transaction time can be improved by almost 6 seconds.

6.2.2 Network Knowledge Enhancement Test

In our Network Knowledge Enhancement Test, we set up multiple networks, which we'll refer to as Network A, Network B, Network C, and so on. Each of these networks had its own dedicated broker and provider, labeled correspondingly (i.e., Network A had Broker A and Provider A). Crucially, Broker A was linked to Provider A within Network A. Subsequently, devices within Network A were also interconnected with those in Network B, and this pattern continued across the networks.

The objective is to evaluate the redirect functionality within the querying system.

Figure 20: Time to response for each case of entrypoint discovery



When a query is executed in Network A with the redirect setting at 1, we anticipate the results to include Provider A, Provider B, and Provider C. Providers A and B are derived directly from the initial query, while Provider C is sourced from the redirected query to Broker B.

We observed that even as a node's knowledge of the network increased, its capability to discover new entities remained unchanged. The underlying reason for this stagnation was that the caching system currently in place only caches providers and does not account for the brokers.

Without brokers being cached, the system's reach in terms of discovery remains limited, regardless of how much more it knows about the existing providers. To address this shortcoming and truly enhance network discoverability, we recommend that future iterations of the system explore mechanisms for caching brokers. Incorporating brokers into the cache could potentially streamline and bolster the querying process, leading to more efficient network utilization.

6.2.3 Query Discovery vs Cache-Assisted Discovery Time Test

To kick things off, a virtual Swarm network has been set up. Within this configuration, the network comprises a node dedicated to the payment and reputation blockchain, a singular consumer, and a range of providers varying between 1 to 5 entities.

During the initial phase, the "Query-Assisted Connection Phase", every node starts

Table 7: Comparison of query vs cache-assisted discovery time

Cameras	Query Discovery(s)	Cache-Assisted Discovery(s)
1	18.88	3.82
2	19.26	4.20
3	19.62	4.55
4	20.00	4.55
5	20.39	5.38

with a clean slate, meaning they have zero prior knowledge of each other. In this context, the consumer initiates a query directed towards the network. Given that the query matches, every provider responds. The transaction duration is delineated from the moment the providers dispatch the initial query to the network and concludes upon the utilization of the service, marking the final feedback's transmission to the blockchain.

Transitioning to the "Cache-Assisted Connection Phase", it's noteworthy that the consumer and providers have each other in the memory.—they're well-acquainted thanks to previous interactions. consumer initiates a query inside its own in memory list and after the result of this query, a direct connection request to every provider, and again, we measure the duration it takes for the whole transaction to finish.

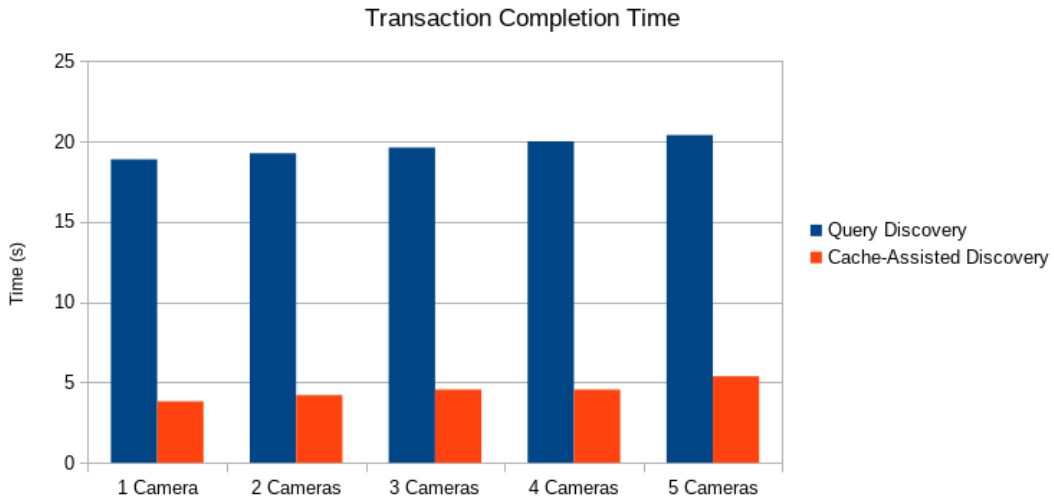
In our analysis of transaction completion times on the Swarm network, we observed some particularly intriguing patterns. Central to our findings was the undeniable advantage that caching conferred upon transaction speeds. When the system utilized cache-assisted discovery methods, the time taken to complete the whole transaction was considerably reduced compared to traditional query-based methods.

The cache, in essence, serves as a memory bank, storing details of previously discovered nodes. This eliminates the need for the system to actively search out nodes within the network for each transaction, as would be the case in a query-based approach. Instead, by leveraging the stored information, nodes can directly communicate with their counterparts, expediting the connection and subsequent transaction process. The data underscores the importance of incorporating caching mechanisms in P2P networks, especially for applications where speed is paramount.

6.2.4 Scalability Analysis

When utilizing a querying system in a Swarm network, each request often requires multiple hops between devices before reaching its intended destination. These intermediary steps can introduce latency and potentially hinder efficient discoverability. However,

Figure 21: Comparition of transaction time from query vs cache-assisted discovery



by integrating caching into the process, the speed for these numerous hops is significantly reduced. As a result, not only is the query speed enhanced, but the overall discoverability within the network also experiences a marked improvement, ensuring more efficient and rapid node-to-node communications.

The limitation observed in the caching system, where only agents are cached and not their brokers, resulted in no enhancement in network knowledge. Identifying this aspect provides valuable insights for future iterations of SwarmOS, highlighting a specific area that can be targeted for improvement.

6.3 Fairness

6.3.1 Reputation Test

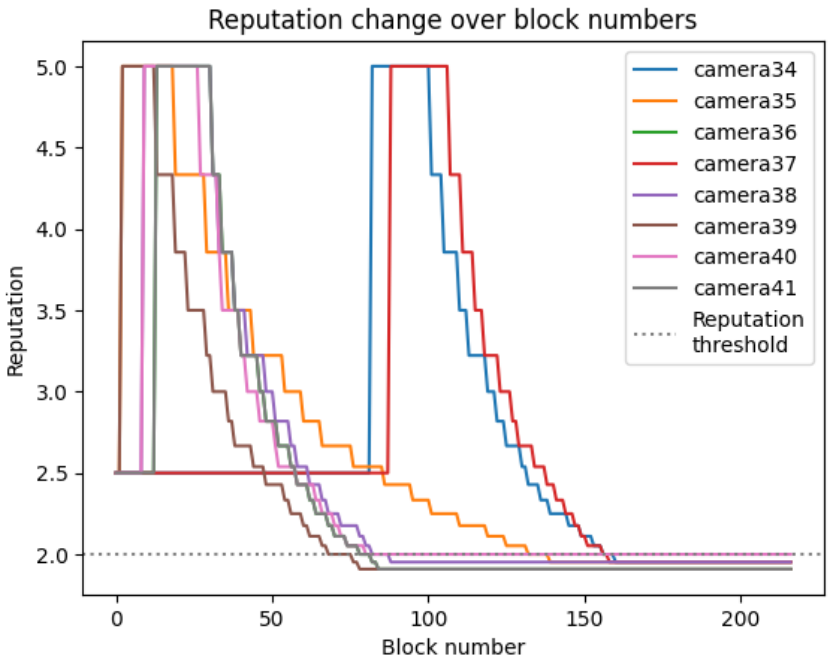
Use cases 1 and 2 focus on a reputation analysis of agents in an environment with default consumers and default providers, and malfunctioning providers.

Use case 1 - Providers with invalid HTTP responses

The simulation of both normal and malfunctioning agents enables a comprehensive evaluation of the reputation system's performance, specifically in terms of its response to malfunctions.

During this phase, cameras numbered from 1 to 32 are considered to be functioning correctly. However, cameras numbered beyond 32 manifest a bug wherein they initially function correctly, but after five requests, they experience crashes, leading to the genera-

Figure 22: Reputation change on use case 1



tion of erroneous HTTP responses.

Figure 22 illustrates this use case in the reputation system, we can see the malfunctioning camera's reputation updating over the block number, where the block number represents the chronological order of the block's inclusion in the reputation blockchain. They slowly start losing reputation, and when their reputation goes down lower than the threshold they stop being requested.

Subsequently, cameras 34 and 37 begin receiving requests as the older cameras are no longer detected in the queries. However, it is observed that these cameras also exhibit the same bug, and after handling some requests, they too get expelled from the network.

Given the structure of the feedback formula explained in section 5.1, the consumer's feedback registers at 0 due to malfunctions, while the broker's feedback is rated at 5, reflecting an error-free and penalty-free transaction from the broker's end.

Using an $\alpha = 0.8$ the reputation score for a malfunctioning request is:

$$Points_i = \alpha F_i + (1 - \alpha) F_b$$

$$Points_i = 0.8 \cdot 0 + (1 - 0.8) \cdot 5 = 1$$

Let's denote 'g' as the count of positive evaluations, and 'b' as the number of unfavorable reviews, then:

$$(5g + 1b)/(g + b) < 2$$

$5g + b < 2g + 2b$

$3g < b$

From these computations, it becomes clear that to effectively remove an unsatisfactory participant from the network, we need thrice the number of negative responses compared to positive ones. This trend holds true both in our formal analysis and simulation studies - it shows that to expel a subpar provider from the network, at least three times more negative experiences are necessary. Even under optimal conditions, where $\alpha = 1$, there needs to be 1.5 times more negative interactions than positive to ensure the removal of an ineffective provider from the network.

Use case 2 - Providers with slow response time

During the evaluation of the reputation system, providers with slow response timers were included in the testing. Figure 23 illustrates a use case scenario where cameras 34 to 41 initially responded normally to requests but experienced a decline in response time after the first 5 interactions.

To assess the impact of delayed responses on the provider's reputation, penalties were applied based on the extent of the response delay. However, the provider's reputation score was never reduced to zero, as long as the responses remained valid.

Moreover, this experimentation showcased the reputation system's capacity to expel malfunctioning agents from the network. The average score for a slow response was about 1.48. Evaluating again on the formula.

$(5n + 1.48k)/(n + k) < 2$

$5n + 1.48k < 2n + 2k$

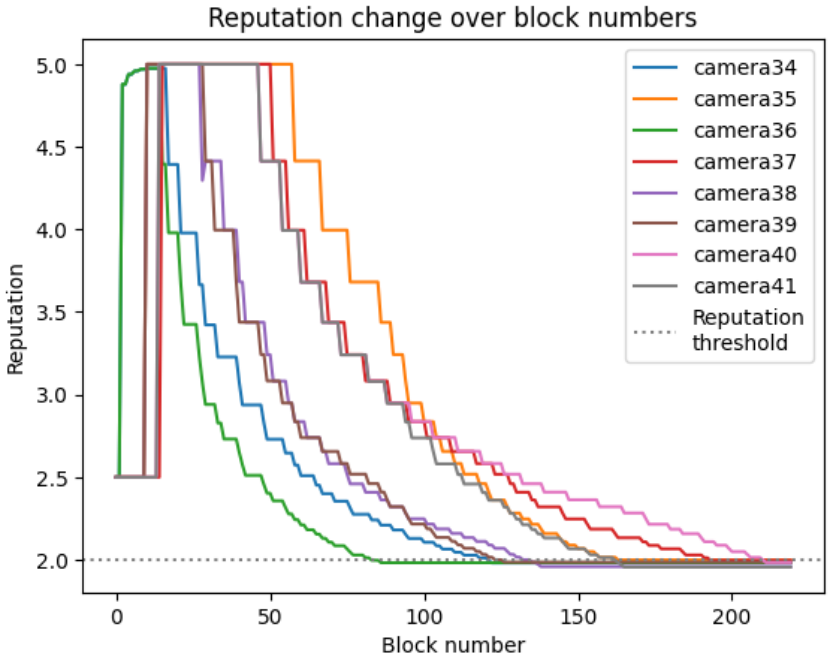
$5.77n < k$

Again, a minimum of 5.77 more malfunctioning requests were needed for expelling a broken provider from the network. The simulation confirms this, requiring on average 28 bad requests to eliminate an agent that had previously provided five good requests.

Use cases 3 and 4 focus on a reputation analysis of agents in an environment with default consumers/providers, together with malicious providers.

Reputation analysis in a simulation with malicious providers can provide insights into how a reputation system behaves in the presence of selfish behavior. Malicious providers can significantly hinder the scalability of a network by disrupting normal operations and

Figure 23: Reputation change on use case 2



causing instability. Their presence can lead to increased communication overhead, as the network needs to constantly identify and handle malicious activities. This added burden can strain the network's resources and impede its ability to efficiently scale. Moreover, malicious providers exploit vulnerabilities in the system, causing cascading failures and reducing the network's overall robustness.

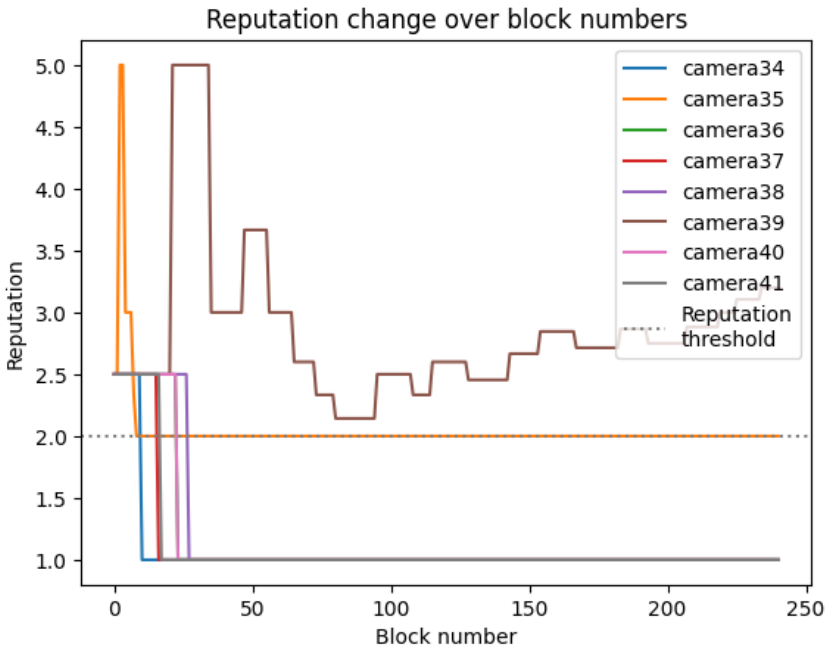
Use case 3 - Providers doing on-off attacks

In a reputation system, a malicious device can attempt to manipulate the system by providing random responses, good or bad, a strategy known as an 'On-off attack'. This is where devices or agents share their observations or experiences about the behavior of other agents. Random responses can distort the reputation scores of the device, potentially impacting its ability to effectively participate within the system.

In Figure 24, we showcase an experiment involving malicious devices. Here, there's an equal probability (50% each) of providing either a good or a bad response. It's notable that all cameras, except camera 39, are expelled from the network as their reputation falls below the predetermined reputation threshold.

The unpredictability of the on-off attack typically leads to the expulsion of most malicious agents from the network during the initial interactions. However, if a malicious agent manages to endure this stage, it's theoretically expected that its score will average around 3.0 (the midpoint between 5.0 and 1.0, the expected reputation score for good and

Figure 24: Reputation change on use case 3



bad responses respectively). Therefore, an on-off attack with a 50%/50% split is already sufficiently disruptive to avert expulsion from the network.

The logical next question is whether these malicious agents could enhance their strategy by initially building a stronger reputation with five good responses before randomly providing bad responses. Theoretically, their reputation score would gravitate towards 3, the average between 5 and 1, comfortably exceeding the expulsion threshold of 2.0. This case can be seen on Figure 24.

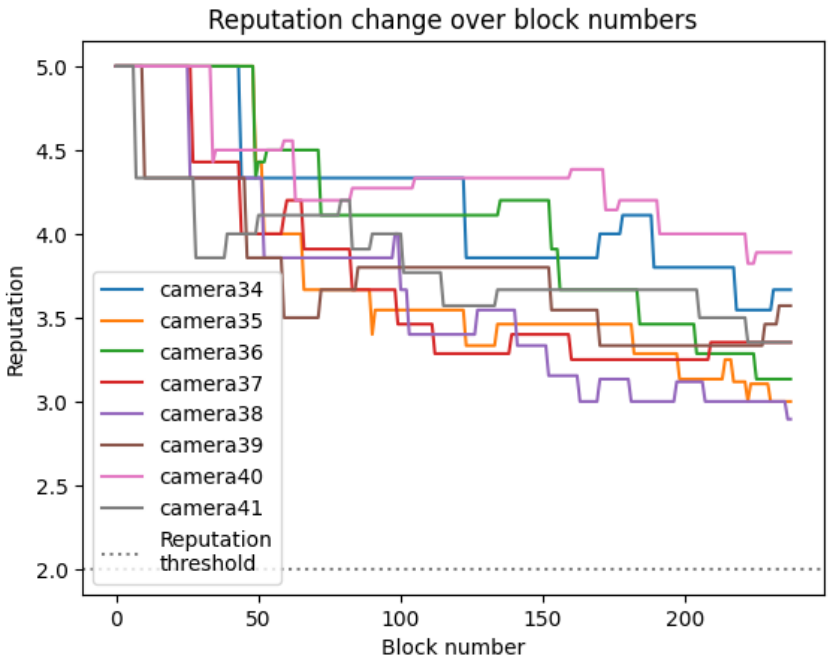
Use case 4 - Providers doing opportunistic on-off attacks

In reputation systems, an "opportunistic attack" is a particular type of malicious behavior in which a device provides false or misleading information only when it's advantageous for the device itself. In these instances, the malicious device cleverly manipulates its reputation score by selectively supplying deceptive data to the system, in a way that optimally benefits the device.

In Figure 24, we explore an experiment involving such opportunistic on-off attacks. These devices operate on a simple strategy: if the reputation falls below 3.0, the agent switches to providing a valid response. Intriguingly, these devices manage to maintain an average reputation score despite frequently supplying misleading responses. Consequently, they continue to remain within the network.

This indicates that even irregular but strategically placed valid responses can be

Figure 25: Reputation change on use case 3 with a initial reputation



sufficient to maintain an acceptable reputation score. Hence, it underscores the need for reputation systems to be designed and implemented in a way that accounts for and mitigates such opportunistic behaviors.

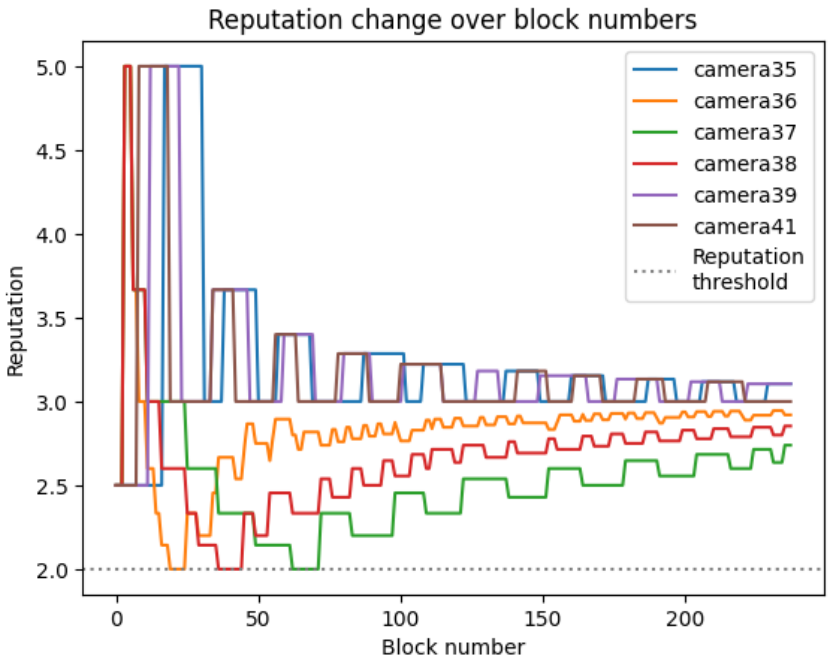
Use cases 5 and 6 focus on analyzing the reputation of gang-backed malicious actors. An enhanced tactic for a malicious entity involves organizing in a coordinated, gang-like fashion. By simulating the actions of malevolent entities operating within such coordinated groups, we can examine their influence on the overall operation of the reputation system and assess its proficiency in detecting and managing such organized malevolent conduct.

Testing the resilience of the reputation system against a range of coordinated malevolent activities, including deception, fraudulent activities, or poor service provision, allows us to evaluate the efficacy of the system's mechanisms in both recognizing and responding to these behaviors. This process can reveal valuable insights into the system's potential vulnerabilities and assist in identifying possible enhancements.

Use case 5 - Gang Attack

In the realm of consumers, strategies often involve self-promotion attacks and bad-mouthing attacks. Regardless of the Quality of Service (QoS), consumers in these scenarios rate gang members highly while providing low scores to other members. This manipulation serves to artificially enhance their reputation while undermining that of ri-

Figure 26: Reputation change on use case 4



vals within the system. Such attacks can be orchestrated by individuals or organizations to inflate their trustworthiness, credibility, or desirability, while simultaneously tarnishing the reputation of competitors. These coordinated endeavors can significantly alter the reputation scores of other agents, potentially impacting their ability to engage within the network. Both self-promotion and bad-mouthing attacks, being malicious in nature, can severely compromise the integrity and accuracy of a reputation system.

Regarding the providers, malicious agents primarily adopt a fraudulent strategy, wherein they accept payment without rendering services to entities outside their gang.

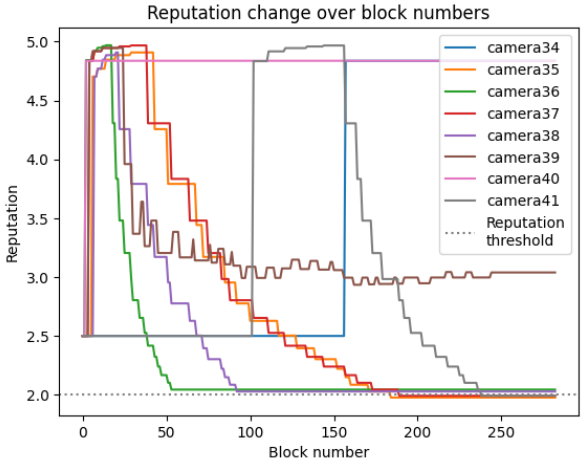
In this analysis, we focus on the variation in the number of consumers. The quantity of providers will not influence the outcome, given the reputation scoring and updating mechanisms remain constant. To influence the reputation system's results, we need to alter the active participants, i.e., the consumers who contribute to the reputation score.

Analyzing the Impact on Coordinated Malicious Providers

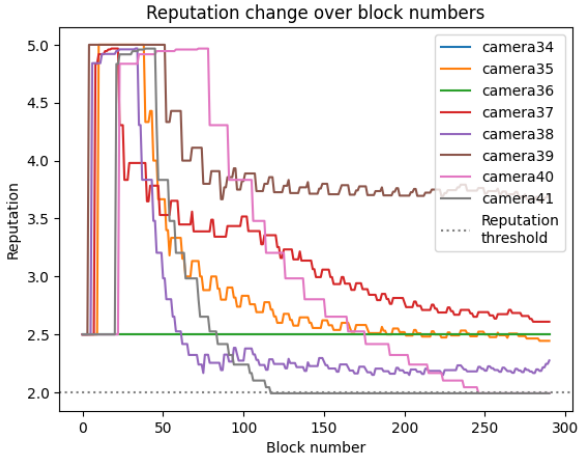
The graph demonstrates the influence of gang formation on reputation scores in networks where 12.5%, 25%, and 50% of consumers belong to gangs. As the proportion of consumers within a gang escalates, their strategy of self-rating favorably becomes increasingly effective. As a result, the graph highlights a direct correlation between the size of the gang and the positive impact on the reputation scores for gang members. The larger the gang, the greater their capacity to manipulate reputation scores in their favor, thereby

Figure 27: Reputation score by block number (Malicious providers)

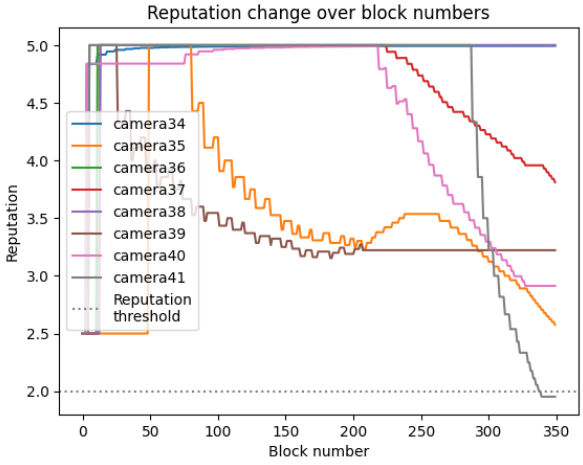
(a) Malicious providers scores on a network with 12.5% of consumers on the same gang



(b) Malicious providers scores on a network with 25% of consumers on the same gang



(c) Malicious providers scores on a network with 50% of consumers on the same gang



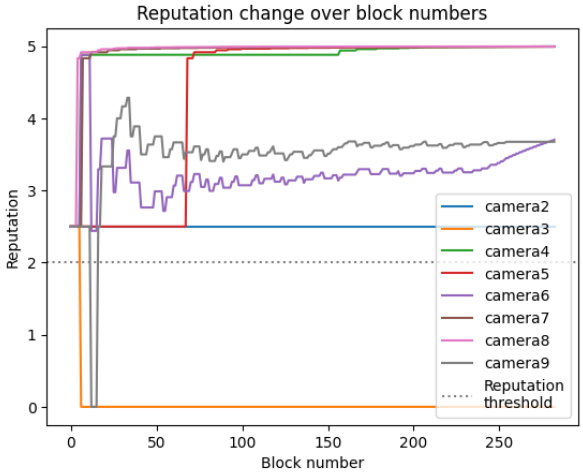
maintaining higher reputations despite their malicious actions.

Analyzing the Impact on Well-Intentioned Providers

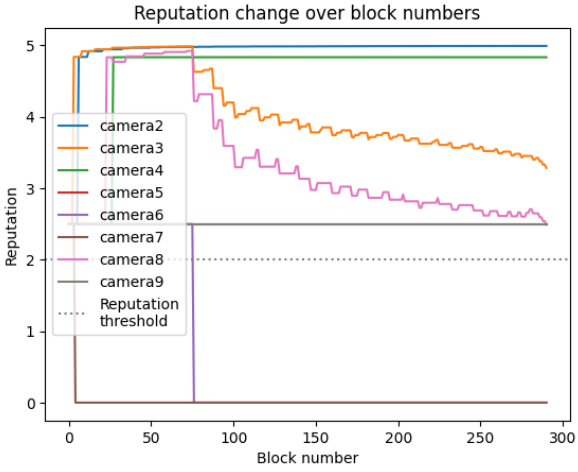
The graph illustrates the predicament of well-intentioned providers who are not part of these gangs. As these providers get systematically sidelined from the gangs, their reputation scores suffer, leading to their eventual expulsion from the network. The graph notably displays how gang-centric dynamics, which involve consumers providing elevated reputation scores to their fellow gang members, foster an environment detrimental to the network participation of well-meaning, non-gang providers. Their reputation scores dip, leading to their systematic exclusion. As the count of consumers within gangs swells, the repercussion of this strategy on the expulsion of well-intentioned providers becomes

Figure 28: Reputation score by block number (Well intentioned providers)

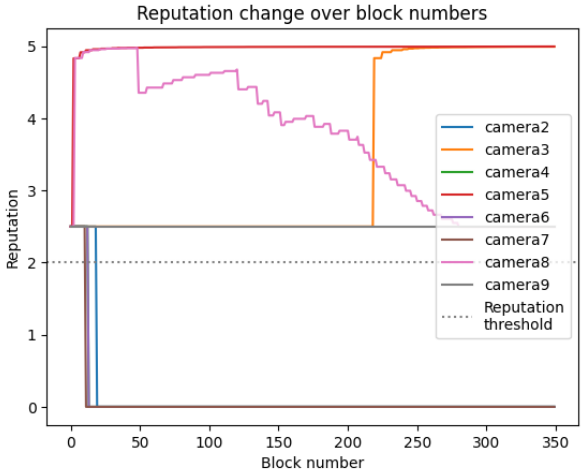
(a) Well intentioned providers scores on a network with 12.5% of consumers on the same gang



(b) Well intentioned providers scores on a network with 25% of consumers on the same gang



(c) Well intentioned providers scores on a network with 50% of consumers on the same gang



glaringly evident. This underlines the substantial influence of gang dynamics on network participation and the distribution of reputation scores.

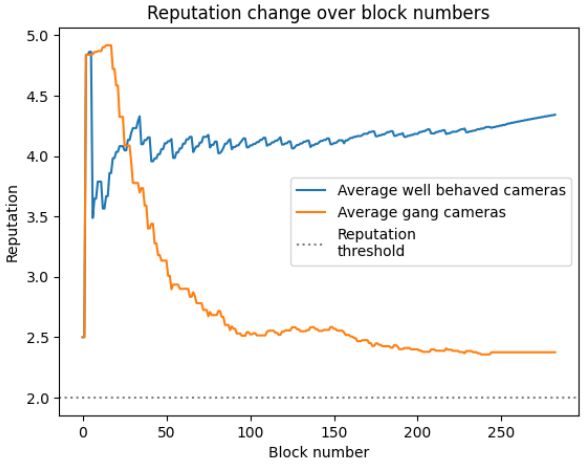
Analyzing the Impact of Gang Membership on Reputation Scores

Subsequently, we altered the proportion of gang members within the network, incrementally increasing their presence from an initial 12.5% to 25% and ultimately to 50%. At each stage, we reevaluated the average reputation scores of both the well-behaved agents and the gang members, providing us with a clear understanding of the effects of increased gang membership on the network.

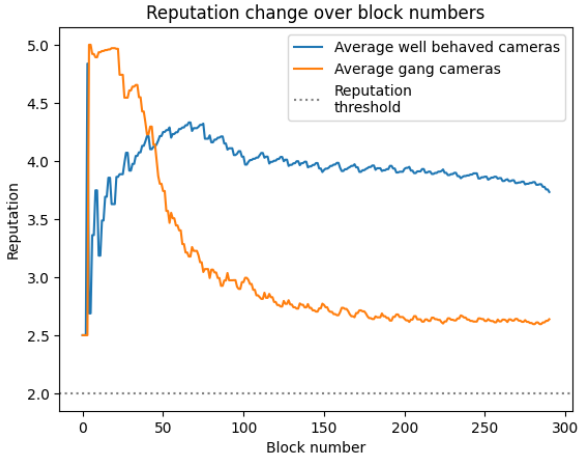
This stepwise escalation allowed us to witness the evolving dynamics within the net-

Figure 29: Use case 5 average reputation score

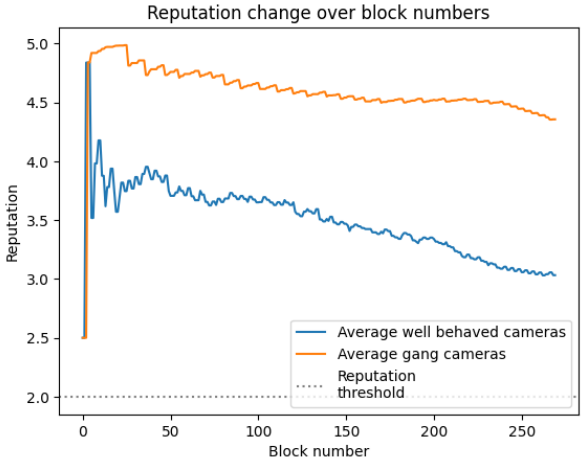
(a) Average score of well-intentioned vs gang cameras with 12.5% of consumers on the same gang



(b) Average score of well-intentioned vs gang cameras with 25% of consumers on the same gang



(c) Average score of well-intentioned vs gang cameras with 50% of consumers on the same gang



work, revealing the pervasive influence of increasing gang membership on reputation scores. The resultant data from this comprehensive study provided valuable insights into how different levels of gang representation can significantly impact the reputation-based dynamics of the network.

In essence, this analysis shines a light on the potential vulnerabilities of reputation-based systems, especially in the face of orchestrated attacks by organized malicious actors or 'gangs'. By thoroughly understanding these dynamics, we are better equipped to develop effective countermeasures to preserve the integrity and fairness of these systems.

Use case 6 - Opportunistic gang attack

In a scenario akin to the prior one, consumers persistently employ strategies such as

self-promotion and bad-mouthing attacks.

Meanwhile, all the malicious providers resort to an opportunistic on-off attack strategy. In this scheme, the malicious devices toggle between delivering good and bad responses, hinging upon their own reputation score. When their score descends below a predetermined threshold, these providers adjust their behavior to appear virtuous. This behavior jeopardizes the reliability of all agents' reputation scores, complicating the task of expelling malicious devices from the network.

Figure 30 presents a scenario in which gang members engage in collusion. They persistently assign each other high reputation scores within the gang while simultaneously delivering bad reputation scores to well-behaved, non-gang members. This artifice poses significant challenges for the reputation system, making it difficult to differentiate between colluding gang members and honest, non-gang participants.

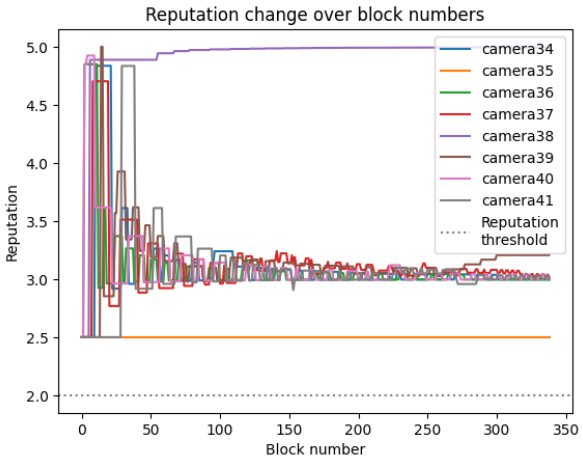
The graphs illustrate how this coordinated effort by gang members to create a positive reputation makes it more arduous to spot their harmful non-functioning devices. It fosters an environment where their malicious actions blend seamlessly with those of the genuine participants. Consequently, the reputation system may encounter hurdles in accurately assessing and differentiating between these two groups. This situation can have a profound impact on the overall accuracy and efficiency of the reputation mechanism, bringing its reliability into question.

The dynamic representation of these graphs sheds light on the resilience of well-intentioned providers who consistently offer legitimate services amidst challenges posed by gang attacks. Despite the strategic maneuvers deployed by these groups, the reputation system demonstrates its effectiveness in identifying and penalizing malicious behavior. These results underscore the system's ability to uphold fairness and transparency, rewarding honest providers for their services while efficiently detecting and countering illicit activities. This resilience positions the reputation system as a robust guardian against external threats, ensuring the integrity of the network and fostering a secure and trustworthy environment for all participants.

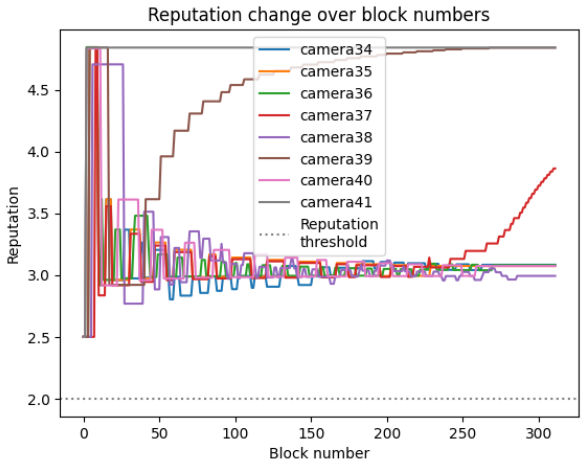
Through comprehensive testing, it became evident that the gangs require a collusion of more than 50% of the consumer network to potentially expel well-intentioned providers. This notable threshold not only emphasizes the robustness of the reputation system but also highlights its effectiveness in safeguarding against collusion attacks. Such a high percentage requirement for collusion serves as a significant deterrent, making it challenging for malicious entities to exploit the system. This aspect, combined with the system's

Figure 30: Reputation score by block number (Malicious providers)

(a) Malicious providers scores on a network with 12.5% of consumers on the same gang



(b) Malicious providers scores on a network with 25% of consumers on the same gang



(c) Malicious providers scores on a network with 50% of consumers on the same gang

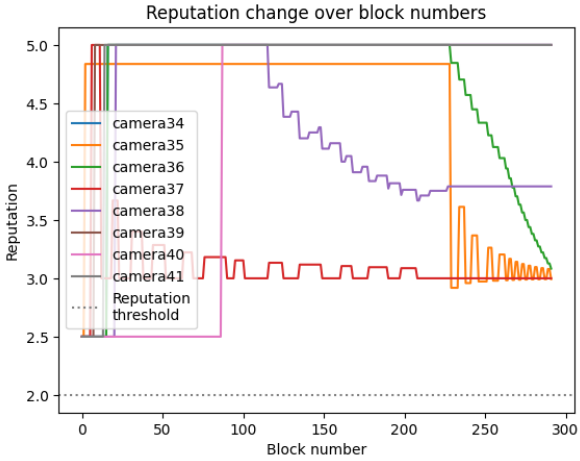
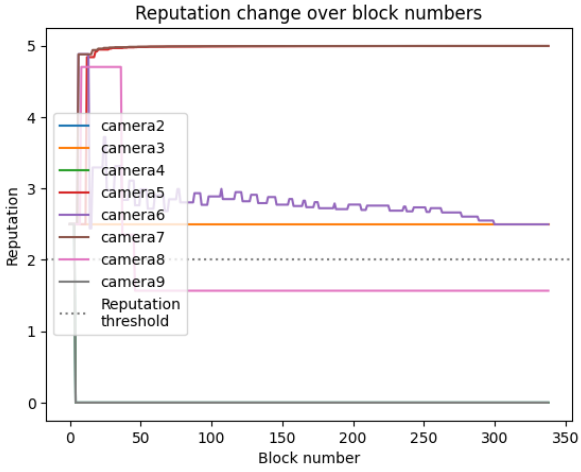
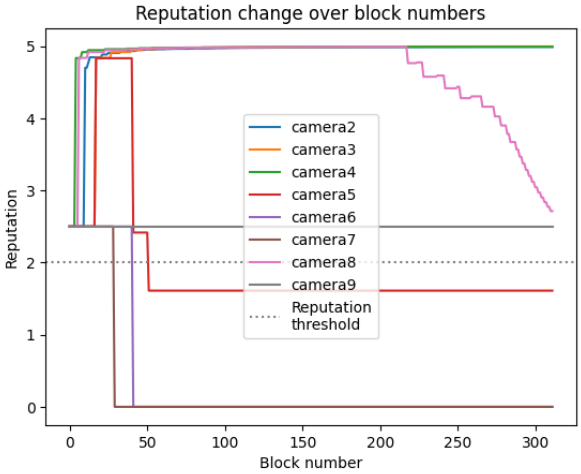


Figure 31: Reputation score by block number (Well intentioned providers)

(a) Well intentioned providers scores on a network with 12.5% of consumers on the same gang



(b) Well intentioned providers scores on a network with 25% of consumers on the same gang



(c) Well intentioned providers scores on a network with 50% of consumers on the same gang

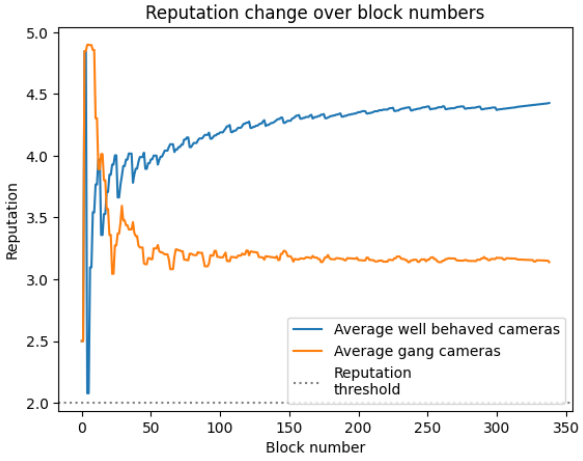
ability to accurately distinguish between good and malicious actors, positions it as a formidable mechanism for maintaining the integrity, security, and fairness of the network.

In a manner akin to the preceding use case, the proportion of gang members within the network was systematically modified, with a gradual increase in their presence from an initial 12.5% to 25%, and eventually reaching 50%.

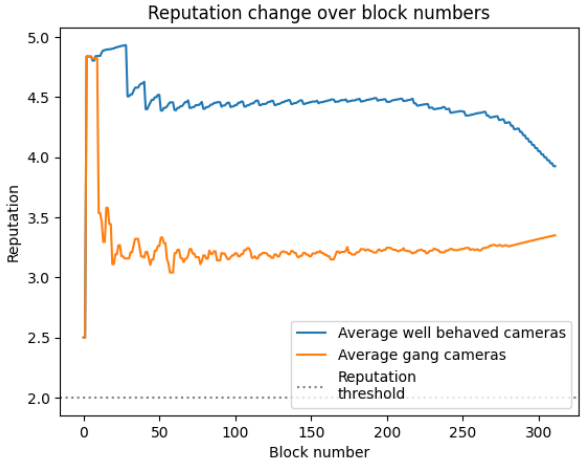
The analysis illustrates the impact that a sufficient number of coordinated gang members can have on a swarm network. The graphs demonstrate how these groups, through strategic coordination, can focus their actions on specific areas inside the network, for example, they can coordinate to attack all known providers of type swarm:camera, effectively dominating these fields.

Figure 32: Use case 6 average reputation score

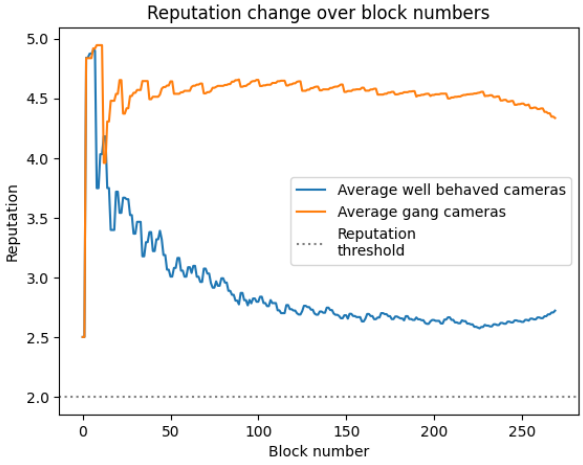
(a) Average score of well-intentioned vs gang cameras with 12.5% of consumers on the same gang



(b) Average score of well-intentioned vs gang cameras with 25% of consumers on the same gang



(c) Average score of well-intentioned vs gang cameras with 50% of consumers on the same gang



Perhaps most crucially, these analyses highlight that while accruing monetary gain,malicious entities monopolize the network. This not only hampers competition but also presents serious threats to network trust, and efficiency.

6.3.2 Scalability Analysis

The reputation system within the network exhibits notable strengths in handling various challenges, as revealed through a detailed scalability analysis.

In the context of malfunctioning providers, the system showcases efficiency by successfully expelling all such entities from the network. While the process may take some time,

the ultimate removal of malfunctioning providers underscores the system's commitment to maintaining a robust and reliable network.

Addressing the issue of malicious providers, the reputation system proves its effectiveness with a basic expulsion strategy. However, the system's sophistication becomes apparent when considering more nuanced strategies employed by malicious actors. Even in the face of these advanced tactics, the reputation system demonstrates resilience, preventing indefinite persistence of malicious providers within the network. This showcases the adaptability and responsiveness of the system to evolving threats.

One of the system's standout features is its ability to combat gang attacks. Extensive testing reveals that these coordinated attacks necessitate collusion among more than 50% of the consumer network to potentially expel well-intentioned providers. This high threshold underscores the robustness of the reputation system in the face of organized efforts to manipulate and compromise network integrity. The system's resistance to such attacks contributes to creating a secure and equitable environment for all providers within the network.

6.4 Privacy

6.4.1 SmartABAC Performance Test

SmartABAC is a novel and expressive ABAC model tailored for autonomous resource-constrained IoT devices. Is also the access control model implemented in SwarmOS.

In the work (FEDRECHESKI et al., 2021) the authors do a comprehensive comparison of execution time across various models together with SmartABAC, the author initially chose models that had existing open-source implementations - namely, PM, HGABAC, and XACML. The comparison was then segmented into three categories: Time to Evaluate Policies, Policy Size, and Lines of Code, providing a multi-faceted analysis of these implementations.

Time to Evaluate Policies: This aspect is a significant consideration within Swarm networks, where evaluations occur for each transaction. The author quantified the time required to assess a single request against the given policies, repeating this process 3000 times for accuracy. This performance testing was executed on a laptop equipped with a 1.8 GHz quad-core processor and 8 GB of memory. The corresponding results are tabulated in Table 8.

Table 8: Comparation of open-source implementations. Reprinted from (FEDRECHESKI et al., 2021)

Criteria	Smart-ABAC		PM	HGABAC	XACML
Time (ms) to run 1 request against policies, p1, p4, and p6, 3000 times	0.18 (C[a])	70 (Elixir[b])	1112 (Ruby[c])	840 (Python[d])	46 (Java[e])
Average policy size (bytes)	85		52	111	3433
Lines of code	449	616	1277	1854	19269

[a] https://github.com/swarm-citi-usp/smart-abac-c
[b] https://github.com/swarm-citi-usp/smart-abac-elixir
[c] https://github.com/mdsol/the_policy_machine
[d] https://github.com/dservos/HGABAC
[e] https://github.com/authzforce/core

Policy Size: Considering the storage and bandwidth limitations often present in IoT devices, it is crucial that embedded access policies are compact. The author conducted a comparative study on policy sizes among various models by gauging their dimensions and calculating the mean. Certain models already possess a predetermined serialization format for policies. For instance, XACML employs Extensible Markup Language (XML), whereas HGABAC utilizes its own policy language, HGPL. These formats were used for expressing and evaluating the respective policies. In the case of SmartABAC, the author utilized CBOR, a condensed yet flexible data format widely adopted in IoT applications. Additionally, CBOR was used to depict the PM graph as an adjacency list, compensating for its lack of an original serialization method.

Lines of code: In addition to performance, the size of a codebase is a consideration for platforms with storage constraints. While the author recognizes that such comparisons depend heavily on the programming language used and the features implemented, it is included for reference. The Linux utility, wc5, was used to count lines, excluding comments and blank lines. Furthermore, given that the PM repository offers various storage adapters, only the in-memory adapter was included in the count for this analysis.

The performance of two SmartABAC implementations were examined across various platforms, as outlined in Table VI. The laptop and Labrador are non-constrained devices operating on Debian 10, while ESP32 and Pulga are MCUs running on bare-metal software. A C-based implementation was assessed across all platforms, whereas the Elixir-based version was only tested on the laptop and Labrador platforms.

Table 9: Performance on different platforms. Reprinted from (FEDRECHESKI et al., 2021)

Description	Platform	SmartABAC (C)	SmartABAC (Elixir)
Time (ms) to run 1 request against 6 policies, 3000 times	Laptop	0.5	76
	Labrador	5.5	830
	ESP32	88	n/a
	Pulga	176	n/a
Time (ms) to run 1 request against 3000 policies	Laptop	0.04	1
	Labrador	0.12	6
	ESP32	1	n/a
	Pulga	4.8	n/a

The test descriptions and their corresponding results are presented in Table VII. In the initial test, the author assessed a single request against the six policies, repeated this process 3000 times, and measured the overall time taken. This scenario simulates an IoT device receiving thousands of authorization requests within a short timeframe. The C implementation exhibited minimal delay on the laptop and Labrador platforms and a tolerable delay on the two resource-constrained platforms. The delay observed when using Elixir on the laptop was also acceptable, although its evaluation delay on Labrador could lead to a noticeable response time delay. In the second test, the author evaluated a scenario of a device in a highly dense network, housing thousands of policies. Here, the author measured the time necessary to evaluate a single request against 3000 policies. The evaluation time in this scenario was negligible for all tested platforms and implementations. Even for the most resource-constrained device, a 32-MHz MCU, less than five milliseconds were required to process 3000 policies.

6.4.2 Scalability Analysis

In conclusion, the scalability analysis of SmartABAC against other open-source models presents a compelling case for its suitability in autonomous resource-constrained IoT devices, particularly within the SwarmOS framework. The analysis was conducted across various dimensions, providing a comprehensive assessment of SmartABAC's performance and efficiency.

The performance results against other open source models are promising, for example, the C implementation of SmartABAC was 255 times faster than its closest competitor, a Java implementation of XACML.

Also the performance test on different platforms shows negligible delay on the laptop

and labrador, and a small delay on more constrained devices considering this test runs 3000 times and measures the total time.

In the second test, it was a device within a densely populated network with thousands of policies. Accordingly, it measured the time required to evaluate a single request against a set of 3000 policies, in this case, every delay was negligible.

While these results provide some indications of our system's capacity to accommodate large networks, actual deployment remains essential to conclusively establish the scalability of the solution.

7 CONCLUSIONS

Scalability is not a straightforward binary concept; it operates along a spectrum with diverse levels of adaptability. In this research it was selected techniques to augment the scalability of the network. Consequently, enhancements have been introduced across various components of SwarmOS.

This research has undertaken a comprehensive analysis of scalability by dissecting it into more accessible and manageable components. By focusing on key elements such as connection, discovery, fairness, and privacy, the study systematically explored how each of these factors influences the overall scalability of the system. This breakdown allowed for a nuanced understanding of the intricate dynamics within each component and provided insights into their collective impact on the system's scalability. Through this approach, the research aimed to unravel the complexities associated with scalability and offer a more granular perspective that can guide targeted improvements in different facets of the system.

The study proposed and specified a set of test cases designed to measure scalability across key elements such as connection, discovery, fairness, and privacy within the Swarm ecosystem. By formulating these test cases, the research aimed to create a structured and systematic framework for assessing the performance and adaptability of the system. This methodology not only allowed for an examination of each scalability aspect but also facilitated the identification of specific strengths and weaknesses within the Swarm environment. Through the careful orchestration of test scenarios, the research sought to provide a comprehensive evaluation that contributes valuable insights into the scalability dynamics of Swarm.

In order to conduct thorough and insightful tests on the scalability of SwarmOS, an aspect of this research involved the development of a robust and flexible testbed implementation. Recognizing the necessity for a tangible and adaptable environment, this research crafted a concrete testbed designed with a high degree of flexibility, ensuring its usability in a diverse array of use cases beyond scalability assessments. The testbed served

as a controlled and dynamic space where various scenarios could be simulated, allowing for the systematic evaluation of SwarmOS performance across different conditions. This deliberate and strategic investment in the development of a versatile testbed underscored the commitment to creating a comprehensive and applicable framework for not only understanding the scalability nuances of SwarmOS but also for potentially informing and enhancing its functionality in real-world applications.

The efficacy of our testbed became evident throughout our work. Even though it was small, it proved invaluable in diagnosing several bugs and areas requiring improvement in SwarmOS. Especially considering that it allowed for efficient identification of shortcomings without necessitating the cost and complexity of vast physical testbeds.

While we take pride in the strides we've made in advancing SwarmOS, this work is aware that the journey of improvement is continuous. Our discoveries and the roadmap for future enhancements will be elaborated upon in the forthcoming 'next steps' section. The horizon of SwarmOS development beckons with promise and potential. Future research directions might include:

1. **Diversifying Network Environments**: While the current research spanned multiple network scenarios, introducing more varied and complex environments could provide even richer data on SwarmOS's performance and resilience.

2. **Enhancing the Caching Mechanism**: The caching mechanism could be refined further, perhaps by introducing machine learning algorithms to predict which endpoints are most likely to be functional based on historical data, further improving transaction times.

3. **Security Considerations**: With the focus on efficiency, it would be worthwhile to assess any potential vulnerabilities introduced by the caching mechanism and design countermeasures to address them.

4. **Integrating Advanced Communication Protocols**: As SwarmOS grows and evolves, it would be beneficial to examine how newer communication strategies or protocols could be amalgamated to boost performance further.

In essence, while our goals of understanding and enhancing SwarmOS's efficiency have been largely achieved, the ever-evolving nature of technology and network systems ensures that this research will remain an ongoing journey, continually adapting and innovating in response to new challenges and discoveries.

This work has ushered in enhancements to the SwarmOS, made possible through the implementation of the **Edge-computing IoT Testing framework**. These improvements are thoroughly documented in the dedicated section 5.4. The testing framework provided a systematic and comprehensive approach to evaluating various aspects of the SwarmOS, leading to targeted refinements. From the scalability of the connection system to the adaptability of brokers in diverse network environments, the testing framework has served as a catalyst for positive changes.

REFERENCES

ADJIH, C. et al. Fit iot-lab: A large scale open experimental iot testbed. In: *2015 IEEE 2nd World Forum on Internet of Things (WF-IoT)*. [S.l.: s.n.], 2015. p. 459–464.

BARBER, S. et al. Bitter to better—how to make bitcoin a better currency. In: SPRINGER. *International conference on financial cryptography and data security*. [S.l.], 2012. p. 399–414.

BELLI, L. et al. Design and deployment of an iot application-oriented testbed. *Computer*, v. 48, n. 9, p. 32–40, Sep 2015. ISSN 1558-0814.

BIASE, L. C. C. D. et al. Swarm economy: A model for transactions in a distributed and organic iot platform. *IEEE Internet of Things Journal*, IEEE, v. 6, n. 3, p. 4561–4572, 2018.

BISWAS, P.; SANDHU, R.; KRISHNAN, R. A comparison of logical-formula and enumerated authorization policy abac models. In: SPRINGER. *Data and Applications Security and Privacy XXX: 30th Annual IFIP WG 11.3 Conference, DBSec 2016, Trento, Italy, July 18-20, 2016. Proceedings 30*. [S.l.], 2016. p. 122–129.

BJERKNES, J. D.; WINFIELD, A. F. On fault tolerance and scalability of swarm robotic systems. In: *Distributed autonomous robotic systems*. [S.l.]: Springer, 2013. p. 431–444.

BOGLIOLO, A. et al. Virtual currency and reputation-based cooperation incentives in user-centric networks. In: IEEE. *2012 8th International Wireless Communications and Mobile Computing Conference (IWCMC)*. [S.l.], 2012. p. 895–900.

BONDI, A. B. Characteristics of scalability and their impact on performance. In: ACM. *Proceedings of the 2nd international workshop on Software and performance*. [S.l.], 2000. p. 195–203.

BORMANN, C.; ERSUE, M.; KERANEN, A. Terminology for constrained-node networks. *Internet Engineering Task Force (IETF): Fremont, CA, USA*, p. 2070–1721, 2014.

BOUCADAIR, M.; PENNO, R.; WING, D. *Universal Plug and Play (UPnP) Internet Gateway Device - Port Control Protocol Interworking Function (IGD-PCP IWF)*. RFC Editor, 2013. RFC 6970. (Request for Comments, 6970). Disponível em: ⟨https://rfc-editor.org/rfc/rfc6970.txt⟩.

BOUCADAIR, M.; PENNO, R.; WING, D. *DHCP Options for the Port Control Protocol (PCP)*. RFC Editor, 2014. RFC 7291. (Request for Comments, 7291). Disponível em: ⟨https://rfc-editor.org/rfc/rfc7291.txt⟩.

BRÖRING, A.; DATTA, S. K.; BONNET, C. A categorization of discovery technologies for the internet of things. In: *Proceedings of the 6th International Conference on the Internet of Things*. [S.l.: s.n.], 2016. p. 131–139.

CABRÉ, J. A. C.; PRECUP, D.; SANZ, R. Horizontal and vertical self-adaptive cloud controller with reward optimization for resource allocation. In: IEEE. *2017 International Conference on Cloud and Autonomic Computing (ICCAC)*. [S.l.], 2017. p. 184–185.

CASALICCHIO, E.; PERCIBALLI, V. Measuring docker performance: What a mess!!! In: *Proceedings of the 8th ACM/SPEC on International Conference on Performance Engineering Companion*. [S.l.: s.n.], 2017. p. 11–16.

CCORI, P. C. et al. Device discovery strategies for the iot. In: IEEE. *2016 IEEE International Symposium on Consumer Electronics (ISCE)*. [S.l.], 2016. p. 97–98.

CHANG, C.; SRIRAMA, S. N.; BUYYA, R. Internet of things (iot) and new computing paradigms. *Fog and edge computing: principles and paradigms*, Springer, v. 6, p. 1–23, 2019.

CHERNYSHEV, M. et al. Internet of things (iot): Research, simulators, and testbeds. *IEEE Internet of Things Journal*, v. 5, n. 3, p. 1637–1647, Jun 2018. ISSN 2327-4662.

CHESHIRE, S. et al. *RFC 6887: Port Control Protocol (PCP)*. [S.l.]: RFC Editor, 2013.

CHESHIRE, S.; KROCHMAL, M. *NAT Port Mapping Protocol (NAT-PMP)*. RFC Editor, 2013. RFC 6886. (Request for Comments, 6886). Disponível em: ⟨https://rfc-editor.org/rfc/rfc6886.txt⟩.

CONOSCENTI, M.; VETRO, A.; MARTIN, J. C. D. Blockchain for the internet of things: A systematic literature review. In: IEEE. *2016 IEEE/ACS 13th International Conference of Computer Systems and Applications (AICCSA)*. [S.l.], 2016. p. 1–6.

COSTA, L. C. P. et al. Swarm os control plane: an architecture proposal for heterogeneous and organic networks. *IEEE Transactions on Consumer Electronics*, v. 61, n. 4, p. 454–462, nov. 2015. ISSN 0098-3063.

COSTA, L. de P. et al. Swarm OS control plane: An architecture proposal for heterogeneous and organic networks. In: *2015 IEEE International Conference on Consumer Electronics (ICCE)*. [S.l.: s.n.], 2015. p. 245–246.

CROMAN, K. et al. On scaling decentralized blockchains. In: SPRINGER. *International conference on financial cryptography and data security*. [S.l.], 2016. p. 106–125.

DIAS, J. P. et al. A brief overview of existing tools for testing the internet-of-things. In: IEEE. *2018 IEEE International Conference on Software Testing, Verification and Validation Workshops (ICSTW)*. [S.l.], 2018. p. 104–109.

FEDRECHESKI, G. et al. Attribute-based access control for the swarm with distributed policy management. *IEEE Transactions on Consumer Electronics*, IEEE, v. 65, n. 1, p. 90–98, 2018.

FEDRECHESKI, G. et al. Smartabac: enabling constrained iot devices to make complex policy-based access control decisions. *IEEE Internet of Things Journal*, IEEE, v. 9, n. 7, p. 5040–5050, 2021.

FORD, B.; SRISURESH, P.; KEGEL, D. Peer-to-peer communication across network address translators. In: *USENIX Annual Technical Conference, General Track*. [S.l.: s.n.], 2005. p. 179–192.

FORTIER, P.; MICHEL, H. *Computer systems performance evaluation and prediction*. [S.l.]: Elsevier, 2003.

FRANKLIN, S.; GRAESSER, A. Is it an agent, or just a program?: A taxonomy for autonomous agents. In: SPRINGER. *International workshop on agent theories, architectures, and languages*. [S.l.], 1996. p. 21–35.

FREEDMAN, M. J.; FREUDENTHAL, E.; MAZIERES, D. Democratizing content publication with coral. In: *NSDI*. [S.l.: s.n.], 2004. v. 4, p. 18–18.

GAŁKA, J.; MASIOR, M.; SALASA, M. Voice authentication embedded solution for secured access control. *IEEE Transactions on Consumer Electronics*, IEEE, v. 60, n. 4, p. 653–661, 2014.

GLOWNIAK, J. History, structure, and function of the internet. In: ELSEVIER. *Seminars in nuclear medicine*. [S.l.], 1998. v. 28, n. 2, p. 135–144.

GUO, J.; CHEN, R.; TSAI, J. J. A survey of trust computation models for service management in internet of things systems. *Computer Communications*, Elsevier, v. 97, p. 1–14, 2017.

GUPTA, A.; CHRISTIE, R.; MANJULA, P. Scalability in internet of things: features, techniques and research challenges. *Int. J. Comput. Intell. Res*, v. 13, n. 7, p. 1617–1627, 2017.

HALES, D. et al. Bittorrent or bitcrunch: Evidence of a credit squeeze in bittorrent? In: IEEE. *2009 18th IEEE International Workshops on Enabling Technologies: Infrastructures for Collaborative Enterprises*. [S.l.], 2009. p. 99–104.

HENDERSON, T. R. et al. Network simulations with the ns-3 simulator. *SIGCOMM demonstration*, v. 14, n. 14, p. 527, 2008.

HU, V. C. et al. Guide to attribute based access control (abac) definition and considerations (draft). *NIST special publication*, Citeseer, v. 800, n. 162, p. 1–54, 2013.

IDC. *The Growth in Connected IoT Devices Is Expected to Generate 79.4ZB of Data in 2025, According to a New IDC Forecast*. 2019. International Data Corporation website. Disponível em: ⟨https://www.idc.com/getdoc.jsp?containerId=prUS45213219⟩. Acesso em: 21 out. 2019.

IETF. *Internet Protocol, Version 6 (IPv6) Specification*. 1998. Request For Comments: 2460. Disponível em: ⟨https://tools.ietf.org/html/rfc2460⟩. Acesso em: 19 mar. 2023.

ISO/IEC 29341 Information technology — UPnP Device Architecture. Geneva, CH, 2017. v. 2017.

KAO, C. H. Survey on evaluation of iot services leveraging virtualization technology. In: *Proceedings of the 2020 5th International Conference on Cloud Computing and Internet of Things*. Association for Computing Machinery, 2020. (CCIOT 2020), p. 26–34. ISBN 978-1-4503-7527-6. Disponível em: ⟨https://doi.org/10.1145/3429523.3429524⟩.

KASH, I. A. et al. Economics of bittorrent communities. In: *Proceedings of the 21st international conference on World Wide Web*. [S.l.: s.n.], 2012. p. 221–230.

KERäNEN, A.; HOLMBERG, C.; ROSENBERG, J. *Interactive Connectivity Establishment (ICE): A Protocol for Network Address Translator (NAT) Traversal*. RFC Editor, 2018. RFC 8445. (Request for Comments, 8445). Disponível em: ⟨https://rfc-editor.org/rfc/rfc8445.txt⟩.

KHATIBI, E.; SHARIFI, M. Resource discovery mechanisms in pure unstructured peer-to-peer systems: a comprehensive survey. *Peer-to-Peer Networking and Applications*, Springer, v. 14, p. 729–746, 2021.

LEE, E. A. et al. The swarm at the edge of the cloud. *IEEE Design & Test*, IEEE, v. 31, n. 3, p. 8–20, 2014.

LEE, E. A. et al. The terraswarm research center (tsrc)(a white paper). *EECS Department, University of California, Berkeley, Tech. Rep. UCB/EECS-2012-207*, 2012.

LEE, J.-H.; KIM, H. Security and privacy challenges in the internet of things [security and privacy matters]. *IEEE Consumer Electronics Magazine*, IEEE, v. 6, n. 3, p. 134–136, 2017.

LIU, Y.; PASSINO, K. M. Swarm intelligence: Literature overview. *Department of electrical engineering, the Ohio State University*, 2000.

LOOGA, V. et al. Mammoth: A massive-scale emulation platform for internet of things. In: IEEE. *2012 IEEE 2nd International Conference on Cloud Computing and Intelligence Systems*. [S.l.], 2012. v. 3, p. 1235–1239.

LY-TRONG, N. et al. Uitiot v3: A hybrid testbed for evaluation of large-scale iot networks. In: *Proceedings of the Ninth International Symposium on Information and Communication Technology*. Association for Computing Machinery, 2018. (SoICT 2018), p. 155–162. ISBN 978-1-4503-6539-0. Disponível em: ⟨https://doi.org/10.1145/3287921.3287935⟩.

MEHDI, K. et al. Cupcarbon: A multi-agent and discrete event wireless sensor network design and simulation tool. In: INSTITUTE FOR COMPUTER SCIENCE, SOCIAL INFORMATICS AND TELECOMMUNICATIONS .... *7th International ICST Conference on Simulation Tools and Techniques, Lisbon, Portugal, 17-19 March 2014*. [S.l.], 2014. p. 126–131.

MENDES, D. X. et al. An experimental reality check on the scaling laws of swarming systems. In: IEEE. *IEEE INFOCOM 2017-IEEE Conference on Computer Communications*. [S.l.], 2017. p. 1–9.

MERRIAM-WEBSTER. *Scalability*. 2021. Disponível em: ⟨https://www.merriam-webster.com/dictionary/scalability⟩.

MISLOVE, A. Post: A secure, resilient, cooperative messaging system. *Notes*, v. 20, p. 22, 2003.

MLOT, N. J.; TOVEY, C. A.; HU, D. L. Fire ants self-assemble into waterproof rafts to survive floods. *Proceedings of the National Academy of Sciences*, National Acad Sciences, v. 108, n. 19, p. 7669–7673, 2011.

MORABITO, R. Virtualization on internet of things edge devices with container technologies: A performance evaluation. *IEEE Access*, v. 5, p. 8835–8850, 2017. ISSN 2169-3536.

MORABITO, R. et al. Evaluating performance of containerized iot services for clustered devices at the network edge. *IEEE Internet of Things Journal*, IEEE, v. 4, n. 4, p. 1019–1030, 2017.

MäKINEN, A.; JIMéNEZ, J.; MORABITO, R. Eliot: Design of an emulated iot platform. In: *2017 IEEE 28th Annual International Symposium on Personal, Indoor, and Mobile Radio Communications (PIMRC)*. [S.l.: s.n.], 2017. p. 1–7. ISSN 2166-9589.

NORDSTRÖM, E. et al. Evaluating wireless multi-hop networks using a combination of simulation, emulation, and real world experiments. In: *Proceedings of the 1st international workshop on System evaluation for mobile platforms*. [S.l.: s.n.], 2007. p. 29–34.

OLSON, K.; WAMPLER, J.; KELLER, E. Doomed to repeat with ipv6? characterization of nat-centric security in soho routers. *ACM Computing Surveys*, ACM New York, NY, 2023.

ÖSTERLIND, F. A sensor network simulator for the contiki os. *SICS Research Report*, Swedish Institute of Computer Science, 2006.

OSTERLIND, F. et al. Cross-level sensor network simulation with cooja. In: IEEE. *Proceedings. 2006 31st IEEE conference on local computer networks*. [S.l.], 2006. p. 641–648.

PIATEK, M. et al. Do incentives build robustness in bittorrent. In: *Proc. of NSDI*. [S.l.: s.n.], 2007. v. 7, p. 4.

POTDAR, A. M. et al. Performance evaluation of docker container and virtual machine. *Procedia Computer Science*, Elsevier, v. 171, p. 1419–1428, 2020.

QUALNET. *Scalable Network Technologies*. 2021. ⟨http://web.scalable-networks.com/qualnet-network-simulator-software⟩. [Online; accessed 25-October-2021].

RAMALHO, F.; NETO, A. Virtualization at the network edge: A performance comparison. In: IEEE. *2016 IEEE 17th International Symposium on A World of Wireless, Mobile and Multimedia Networks (WoWMoM)*. [S.l.], 2016. p. 1–6.

RAMASUBRAMANIAN, V.; SIRER, E. G. The design and implementation of a next generation name service for the internet. *ACM SIGCOMM Computer Communication Review*, ACM New York, NY, USA, v. 34, n. 4, p. 331–342, 2004.

RAMPRASAD, B. et al. Emu-iot - a virtual internet of things lab. In: *2019 IEEE International Conference on Autonomic Computing (ICAC)*. [S.l.: s.n.], 2019. p. 73–83. ISSN 2474-0756.

REDDY.K, T. et al. *Traversal Using Relays around NAT (TURN): Relay Extensions to Session Traversal Utilities for NAT (STUN)*. RFC Editor, 2020. RFC 8656. (Request for Comments, 8656). Disponível em: ⟨https://rfc-editor.org/rfc/rfc8656.txt⟩.

RHEA, S. et al. Handling churn in a dht. In: REPORT NO. UCB/CSD-03-1299, UNIVERSITY OF CALIFORNIA, ALSO PROC. USENIX .... [S.l.], 2003.

ROSENBERG, J. et al. *STUN - Simple Traversal of User Datagram Protocol (UDP) Through Network Address Translators (NATs)*. RFC Editor, 2003. RFC 3489. (Request for Comments, 3489). Disponível em: ⟨https://rfc-editor.org/rfc/rfc3489.txt⟩.

ROSENBERG, J. et al. *RFC 6544: TCP Candidates with Interactive Connectivity Establishment (ICE)*. [S.l.]: RFC Editor, 2012.

ROWSTRON, A.; DRUSCHEL, P. Pastry: Scalable, decentralized object location, and routing for large-scale peer-to-peer systems. In: SPRINGER. *Middleware 2001: IFIP/ACM International Conference on Distributed Systems Platforms Heidelberg, Germany, November 12–16, 2001 Proceedings 2*. [S.l.], 2001. p. 329–350.

RUNTIME options with memory, cpus, and gpus. 2021. Disponível em: ⟨https://docs.docker.com/config/containers/resource_constraints/⟩.

SANCHEZ, L. et al. Smartsantander: Iot experimentation over a smart city testbed. *Computer Networks*, v. 61, p. 217–238, Mar 2014. ISSN 1389-1286.

SARKAR, C. et al. Diat: A scalable distributed architecture for iot. *IEEE Internet of Things journal*, IEEE, v. 2, n. 3, p. 230–239, 2014.

SARMADY, S. A survey on peer-to-peer and dht. *arXiv preprint arXiv:1006.4708*, 2010.

Seetharami Seelam. *How to run more than 1024 docker containers on my system?* 2023. Disponível em: ⟨http://sseelam.blogspot.com/2015/10/how-to-run-more-than-1024-docker.html⟩.

SEHGAL, A. et al. Management of resource constrained devices in the internet of things. *IEEE Communications Magazine*, IEEE, v. 50, n. 12, p. 144–149, 2012.

SILVA, E. d. S. e et al. On the scalability of p2p swarming systems. *Computer Networks*, Elsevier, v. 151, p. 93–113, 2019.

SOTIRIADIS, S. et al. Elastic load balancing for dynamic virtual machine reconfiguration based on vertical and horizontal scaling. *IEEE Transactions on Services Computing*, IEEE, v. 12, n. 2, p. 319–334, 2016.

STANKOVIC, J. A. Research directions for the internet of things. *IEEE internet of things journal*, IEEE, v. 1, n. 1, p. 3–9, 2014.

STOICA, I. et al. Chord: A scalable peer-to-peer lookup service for internet applications. *ACM SIGCOMM computer communication review*, ACM New York, NY, USA, v. 31, n. 4, p. 149–160, 2001.

STUTZBACH, D.; ZAPPALA, D.; REJAIE, R. The scalability of swarming peer-to-peer content delivery. In: SPRINGER. *International Conference on Research in Networking*. [S.l.], 2005. p. 15–26.

TERANISHI, Y. et al. Jose: An open testbed for field trials of large-scale iot services. *Journal of the National Institute of Information and Communications Technology*, National Institute of Information and Communications Technology, v. 62, n. 2, p. 151–159, 2016.

TO, M. A.; CANO, M.; BIBA, P. Dockemu–a network emulation tool. In: IEEE. *2015 IEEE 29th international conference on advanced information networking and applications workshops*. [S.l.], 2015. p. 593–598.

TOBIN, A.; REED, D. The inevitable rise of self-sovereign identity. *The Sovrin Foundation*, v. 29, n. 2016, p. 18, 2016.

VARGA, A.; HORNIG, R. An overview of the omnet++ simulation environment. In: *Proceedings of the 1st international conference on Simulation tools and techniques for communications, networks and systems & workshops*. [S.l.: s.n.], 2008. p. 1–10.

WANG, Y.; LU, Z.; GU, J. Research on symmetric nat traversal in p2p applications. In: IEEE. *2006 International Multi-Conference on Computing in the Global Information Technology-(ICCGI'06)*. [S.l.], 2006. p. 59–59.

WANG, Y. et al. An automatic physical access control system based on hand vein biometric identification. *IEEE Transactions on Consumer Electronics*, IEEE, v. 61, n. 3, p. 320–327, 2015.

WHITE, J.; PILBEAM, A. A survey of virtualization technologies with performance testing. *arXiv preprint arXiv:1010.3233*, 2010.

WIKIMEDIA. *Full Cone NAT*. 2020. File: Full_Cone_NAT.svg. Disponível em: ⟨https://en.wikipedia.org/wiki/File:Full_Cone_NAT.svg⟩.

WIKIMEDIA. *NAT IP Address Swapping*. 2020. File: NAT_Concept-en.svg. Disponível em: ⟨https://commons.wikimedia.org/wiki/File:NAT_Concept-en.svg⟩.

WIKIMEDIA. *Port Restricted Cone NAT*. 2020. File: Port_Restricted_Cone_NAT.svg. Disponível em: ⟨https://en.wikipedia.org/wiki/File:Port_Restricted_Cone_NAT.svg⟩.

WIKIMEDIA. *Restricted Cone NAT*. 2020. File: Restricted_Cone_NAT.svg. Disponível em: ⟨https://en.wikipedia.org/wiki/File:Restricted_Cone_NAT.svg⟩.

WIKIMEDIA. *Symmetric NAT*. 2020. File: Symmetric_NAT.svg. Disponível em: ⟨https://en.wikipedia.org/wiki/File:Symmetric_NAT.svg⟩.

WÖRNER, D.; BOMHARD, T. von. When your sensor earns money: exchanging data for cash with bitcoin. In: *Proceedings of the 2014 ACM International Joint Conference on Pervasive and Ubiquitous Computing: Adjunct Publication*. [S.l.: s.n.], 2014. p. 295–298.

WU, D. et al. Understanding peer exchange in bittorrent systems. In: IEEE. *2010 IEEE Tenth International Conference on Peer-to-Peer Computing (P2P)*. [S.l.], 2010. p. 1–8.

WU, J.; LIANG, Q.; BERTINO, E. Improving scalability of software cloud for composite web services. In: IEEE. *2009 IEEE International Conference on Cloud Computing*. [S.l.], 2009. p. 143–146.

ZHOU, X. et al. Ipv6 delay and loss performance evolution. *International Journal of Communication Systems*, Wiley Online Library, v. 21, n. 6, p. 643–663, 2008.

ZYSKIND, G.; NATHAN, O.; PENTLAND, A. Enigma: Decentralized computation platform with guaranteed privacy. *arXiv preprint arXiv:1506.03471*, 2015.

APPENDIX A – PUBLICATIONS

This chapter lists the research articles published by the author during the development of the research.

A.1 Main Articles

The following articles have direct impact on the work presented on this dissertation.

PEREIRA, William T. et al. A virtualized testbed for IoT: Scalability for swarm application. In: **2023 IEEE Consumer Communications & Networking Conference (CCNC)**. IEEE, 2023. p. 1074-1079.

A.2 Other Publications

The following publications were co-authored during the development of this dissertation, and represent indirect results of this research.

FEDRECHESKI, Geovane et al. Self-sovereign identity for IoT environments: a perspective. In: **2020 Global Internet of Things Summit (GIoTS)**. IEEE, 2020. p. 1-6.

AFZAL, Samira et al. Analysis of Web-Based IoT through Heterogeneous Networks: Swarm Computing over LoRaWAN. **Sensors**, v. 22, n. 2, p. 664, 2022.