

# Decentralized Task Coordination in Heterogeneous IoT Clusters

DIPLOMARBEIT

zur Erlangung des akademischen Grades

**Diplom-Ingenieur**

im Rahmen des Studiums

**Software Engineering/Internet Computing**

eingereicht von

**Filip Loisel, BSc**

Matrikelnummer 00371481

an der Fakultät für Informatik

der Technischen Universität Wien

Betreuung: Univ. Prof. Dr. Schahram Dustdar

Mitwirkung: Dr. Andrea Morichetta

Wien, 3. Mai 2023

---

Filip Loisel

---

Schahram Dustdar



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar  
The approved original version of this thesis is available in print at TU Wien Bibliothek.

# Decentralized Task Coordination in Heterogeneous IoT Clusters

DIPLOMA THESIS

submitted in partial fulfillment of the requirements for the degree of

**Diplom-Ingenieur**

in

**Software Engineering & Internet Computing**

by

**Filip Loisel, BSc**

Registration Number 00371481

to the Faculty of Informatics

at the TU Wien

Advisor: Univ. Prof. Dr. Schahram Dustdar

Assistance: Dr. Andrea Morichetta

Vienna, 3<sup>rd</sup> May, 2023

---

Filip Loisel

---

Schahram Dustdar



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar  
The approved original version of this thesis is available in print at TU Wien Bibliothek.

# Erklärung zur Verfassung der Arbeit

Filip Loisel, BSc

Hiermit erkläre ich, dass ich diese Arbeit selbständig verfasst habe, dass ich die verwendeten Quellen und Hilfsmittel vollständig angegeben habe und dass ich die Stellen der Arbeit – einschließlich Tabellen, Karten und Abbildungen –, die anderen Werken oder dem Internet im Wortlaut oder dem Sinn nach entnommen sind, auf jeden Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht habe.

Wien, 3. Mai 2023

---

Filip Loisel



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar  
The approved original version of this thesis is available in print at TU Wien Bibliothek.

# Danksagung

Ich möchte mich bei meinen Betreuern, Univ. Prof. Dr. Schahram Dustdar und Dr. Andrea Morichetta, für ihre vielseitige Unterstützung bedanken. Durch Andrea's wegweisenden Ratschläge und Beiträge konnte ich die Master Arbeit nach meinen eigenen Vorstellungen formen. Die Arbeit mit ihm war stets professionell und auf Augenhöhe. Außerdem möchte ich mit bei den anderen Kollegen aus der Distributed Systems Group, besonderes Alexander Knoll und Philipp Raith, für die Unterstützung bedanken.

Mein Dank geht auch an meinen guten Freund und Kollegen, Geri Zeqo. Zusammen haben wir uns gegenseitig positiv herausgefordert und jegliche Probleme und Schwierigkeiten im Dialog meistern können.

Zu guter Letzt, möchte ich mich herzlich bei meiner Familie, im Besonderen bei meiner Mutter bedanken. Sie hat mich nicht nur während dem Schaffen dieser Master Arbeit unterstützt, sondern während meines ganzen Studiums. Sie hat mir gezeigt, dass der wichtigste Aspekt eines Software Engineers, nicht die technische, sondern die menschliche Seite ist. Schlussendlich sind wir dafür verantwortlich, Software für Menschen zu bauen.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar  
The approved original version of this thesis is available in print at TU Wien Bibliothek.

# Acknowledgements

I would first like to thank my thesis advisors, Univ. Prof. Dr. Schahram Dustdar and Dr. Andrea Morichetta, for their versatile support. Working with Andrea was always a professional and eye-to-eye experience. Through his guiding advice and contributions, I was able to shape the Master's thesis according to my own ideas. Furthermore, I would like to thank my other colleagues from the Distributed Systems Group, especially Alexander Knoll and Philipp Raith, for their support.

My thanks also go to my friend and colleague, Geri Zeqo. We positively challenged each other and were able to master any technical problems and difficulties in cooperation with each other.

Last but not least, I would like to express my sincere gratitude to my family, especially to my mother. She has supported me not only during the creation of this Master's thesis but throughout my studies. She showed me that the most critical aspect of a software engineer is not the technical side but the human side. In the end, we are in charge of building software for people.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar  
The approved original version of this thesis is available in print at TU Wien Bibliothek.

# Kurzfassung

Die ständige Ausbreitung von IoT Geräten in verschiedenen Industrien und die steigende Komplexität solcher intelligenten Systeme, fordert ein Umdenken der Abhängigkeit von Ressourcen in der Cloud zu einer dezentralen IoT Architektur. Im neuen Modell kollaborieren IoT Geräte lokal untereinander um weiterentfernte Datenzentren, Latenz und Bandbreite zu entlasten.

Dieser Paradigmenwechsel wirft neue Forschungsfragen auf, wie die der dezentralen Koordination von Arbeitslast. IoT Geräte formen sich selbständig zu heterogenen Clustern in denen sich sich untereinander Tasks zuteilen können. In dieser Arbeit betrachten wir, wie diese Zuteilung dezentral zur Laufzeit aussehen kann. Die Koordination von Aufgaben muss die verfügbaren Ressourcen und Kapazitäten im Cluster beachten. Außerdem darf sie keine statische Infrastruktur wegen der dynamischen Natur von IoT Geräten annehmen.

Zu diesem Zweck haben wir insgesamt drei Task-Koordinierungsstrategien entworfen und entwickelt. Wir verwenden als Ausgangspunkt einen Ansatz der auf Zufall entscheidet, welches Gerät die Arbeit erhält und verwenden diese Strategie als Vergleichsbasis für die restlichen zwei Algorithmen. Die zweite Strategie basiert auf Ant Colony Optimization (ACO) und wurde an unsere Problemstellung angepasst, sodass Pheromone entlang der Verbindungen gelegt werden, die aussagen wie attraktiv ein Pfad ist. Der dritte und letzte Algorithmus ist mit einem einfachen Gossip-Protokoll gebaut worden.

Wir haben eine Vielzahl an Experimenten mit 10, 25, 50 und 100 Geräten (VMs) in einer statischen und dynamischen Umgebung mit Geräteausfällen durchgeführt. Wir können zeigen, dass ACO mit der kleinsten Anzahl an Hops und verschickten Nachrichten einen passenden Knoten findet, während die Gossips Strategie am meisten Tasks erfolgreich zuweisen kann. Gossips zeigt damit die beste Systemverlässlichkeit, dafür skaliert ACO besser. Die Arbeit hebt hervor, dass ACO besser abschneidet als die Vergleichsunterlage mit dem Zufallsansatz.

Wir können folgern, dass ACO sich als vielversprechender Kandidat für dezentrale Taskkoordination in IoT-Clustern zeigt. Unsere Arbeit stellt das Fundament für weitere Forschungen und Verbesserungen in diesem Bereich bereit.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar  
The approved original version of this thesis is available in print at TU Wien Bibliothek.

# Abstract

The continuous expansion of IoT devices in various industries and the increasing complexity of such intelligent systems demand a shift from the cloud-centric model to a decentralized IoT architecture. In the new model, IoT devices collaborate locally with each other to relieve remote data centers, latency, and bandwidth.

This paradigm shift raises novel research questions, such as decentralized workload and task coordination. IoT devices are autonomously forming heterogeneous clusters in which tasks can be assigned to each other. This thesis considers how this allocation can be decentralized at runtime. The coordination of tasks must consider the available resources and capacities in the cluster. Moreover, it must not assume a static infrastructure because of the dynamic nature of IoT devices.

To this end, we have designed and developed three task coordination strategies. As a starting point, we use an approach that decides on randomness which device gets the work, and employ this strategy as a comparison baseline for the remaining two algorithms. The second algorithm is based on Ant Colony Optimization (ACO) and was adapted to our problem so that pheromones are placed along the links that tell how attractive a path is. The third and final strategy was built using a simple gossip protocol.

We have conducted various experiments with 10, 25, 50, and 100 devices (VMs) performed in a static and dynamic environment with device outages. We show that ACO finds a matching node with the smallest number of hops and messages sent, while the Gossips strategy can allocate the most tasks successfully. Gossips thus shows the best system reliability, but ACO scales better. The work highlights that ACO performs better than the baseline approach Random.

ACO demonstrates a promising candidate for decentralized task coordination in IoT clusters. Our work provides the foundation for further research and advancements in this area.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar  
The approved original version of this thesis is available in print at TU Wien Bibliothek.

# Contents

<b>Kurzfassung</b>	<b>xi</b>
<b>Abstract</b>	<b>xiii</b>
<b>Contents</b>	<b>xv</b>
<b>1 Introduction</b>	<b>1</b>
<b>2 Background</b>	<b>7</b>
2.1 The New Decentralized IoT Model . . . . .	7
2.2 Ant Colony Optimization . . . . .	10
2.3 Gossip Protocol . . . . .	12
<b>3 Related Work</b>	<b>15</b>
3.1 Decentralized IoT Model . . . . .	15
3.2 Adjacent Thesis . . . . .	17
3.3 ACO-related Work . . . . .	17
<b>4 Methodology</b>	<b>23</b>
4.1 Requirements & Objectives . . . . .	23
4.2 System Architecture . . . . .	25
4.3 Task Coordination Strategies . . . . .	30
4.4 Communication . . . . .	44
4.5 System Monitoring . . . . .	46
<b>5 Evaluation</b>	<b>49</b>
5.1 Experiments Setup . . . . .	49
5.2 Testbed . . . . .	54
5.3 Results . . . . .	55
<b>6 Discussion</b>	<b>71</b>
6.1 Comparison of Strategies . . . . .	71
6.2 Use Cases . . . . .	73
6.3 Limitations . . . . .	75
	xv

<b>7 Conclusion</b>	<b>77</b>
7.1 Future Work . . . . .	78
<b>List of Figures</b>	<b>81</b>
<b>List of Tables</b>	<b>83</b>
<b>List of Algorithms</b>	<b>85</b>
<b>Glossary</b>	<b>87</b>
<b>Acronyms</b>	<b>89</b>
<b>Bibliography</b>	<b>91</b>
<b>Appendix</b>	<b>95</b>
Gitlab Repositories . . . . .	95

# Introduction

With the proliferation of connected Internet-of-Things (IoT) devices, we are seeing a steady growth of massive volumes of data generated that will significantly impact our society and economy. New business areas and opportunities are emerging to improve people's quality of life.

IoT devices, usually low-resource nodes equipped with various sensors, appear in more and more areas of life. These devices gather data in hospitals, cities, factories, etc. This data is then collected, processed, and analyzed for decision-making purposes and advancements. For this reason, it makes sense that the number of IoT devices is skyrocketing: The International Data Corporation (IDC) predicts that 41.6 billion devices will be in use by the end of 2025 – capable of generating 79.4 zettabytes of data [10]. Furthermore, McKinsey estimates that by the same time, IoT devices have the potential to produce globally up to 11.1 trillion USD in economic value [15].

The sheer number of IoT devices and the massive amount of data generated poses significant challenges to the current cloud-centric IoT model: The current centralized architecture where requests and data are sent from IoT devices to the cloud is becoming a bottleneck over time:

- **Increased latency.** Cloud and data centers are located far from IoT devices, causing higher latency when transferring data. It complicates the implementation of real-time decisions, such as in the field of autonomous cars and drones.
- **Increased bandwidth.** Furthermore, increased network bandwidth occurs when tens of millions of IoT devices send and receive information to the cloud. This burden of data upload toward data centers also leads to an increased energy consumption.
- **Vulnerable to failures.** Another area for improvement of the current IoT model is that the cloud is a single point of failure, making it less resilient to outages.

- **Privacy concerns.** Additionally, it is important to remember that central clouds impose privacy issues. IoT devices have little control over what data is processed and to whom it is sent.
- **Inefficient use of resources.** Last but not least, the resources of IoT devices can be better utilized if these nodes search for available services in the surrounding environment instead of contacting the cloud.

For the reasons mentioned above, the current cloud-centric architecture must be reviewed. I.e., a paradigm shift must happen. An alternative to the current model is to bring resources closer to IoT devices. Edge and Fog computing can help balance the bottlenecks. Another step is to make existing IoT and edge devices more self-reliant. They can be seen as self-managing entities that autonomously make decisions and use local resources independently. Especially in an IoT setting, where a multitude of heterogeneous devices that are resource-constrained exists, failures can often occur. IoT nodes leave and join networks at arbitrary times. Mobility and dynamic behavior exacerbate this effect. For these reasons, the approach that IoT devices form local clusters or swarms is a viable alternative to the cloud-centric paradigm. With the decentralized model, devices can assist each other and use resources locally. It reduces latency and bandwidth, improves flexibility to failures due to the dynamic nature, and moves data processing control closer to the data owner. However, this new decentralized model of IoT swarms brings new challenges. Namely, how can heterogeneous devices communicate with each other, and how can they schedule the coordination of tasks with limited knowledge of the available resources of the other swarm devices – taking into account that these nodes may fail or new resources may be added at any given time? This master thesis considers the problem of decentralized task coordination and seeks to find a solution that can be applied in a self-managing, self-learning, and self-adapting cluster. The goal is to master distributed collaborations between heterogeneous devices.

The master thesis aims to design, implement, test, and evaluate a strategy for decentralized task coordination between heterogeneous IoT devices in a dynamic cluster with node dropouts. Each IoT device has its tasks and a queue. When this queue is full, it tries to find other IoT devices in the cluster that 1.) support this task and 2.) are not busy themselves. If a device can be found, this task is offloaded. This type of task coordination is challenging because it cannot be statically defined. Due to IoT devices' dynamic and heterogeneous nature, any device can fail and rejoin the cluster at any time. For this reason, we have chosen a metaheuristic optimization algorithm and adapted it for resource-constrained IoT devices.

The algorithm is heavily based on Ant Colony Optimisation (ACO). ACO is a probabilistic approach and can be reduced to finding a good path in the graph. A fixed number of ants wander through the graph and evaluate the best path based on the edges' weights by leaving pheromones on the edges for other ants to orient. This procedure is repeated in several iterations. Afterward, the paths are compared, and the best one is taken. We have adapted the strategy to our IoT use case to minimize the number of messages in

the system so that task coordination does not involve too much overhead for the already resource-constrained devices. Moreover, the approach considers the node's capacities and reliability. Therefore, the cluster can learn how to adapt with time. We compare our algorithm with a Random strategy and a Gossips-based method to evaluate it with our algorithm. In the Random approach, the task request is randomly forwarded from one node to another until a device can support this task type and still has enough capacity. For the latter strategy, we use the technique of sending messages to neighbors called Gossips, which send task requests to the next neighbors until a node is found.

The research questions that this thesis aims to address are:

1. **RQ1.** How to efficiently coordinate tasks in a dynamic, resource-constrained, and decentralized IoT setting, where nodes can fail, leave, and rejoin the network at any time?
2. **RQ2.** How does the decentralized Ant Colony Optimization-based algorithm impact the task coordination performance compared to the Random and Gossips-based approaches?
3. **RQ3.** How does the dynamic nature of heterogeneous IoT devices in a cluster impact the task coordination performance of the decentralized algorithms?

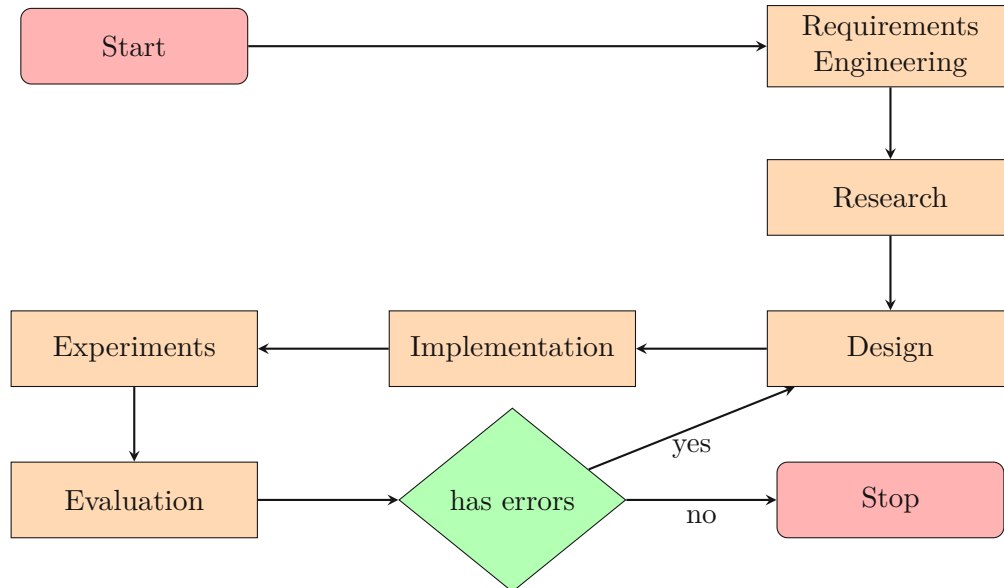


Figure 1.1: Flow chart showing the various stages performed during the master thesis.

The work of this Master's thesis contains several parts in order to be able to answer the research questions: First, we perform a requirements engineering phase to define the exact goals of this thesis. It helps us limit the work, draw boundaries, identify requirements, and set priorities. After that, a research phase takes place in which we look for possible solutions for decentralized task coordination in the IoT domain. We narrowed down our search for ACO and tried finding distributed ACO alternatives. Once the strategies were defined, we defined them in detail and implemented them as a framework. The goal is that IoT devices can use this framework to specify their tasks and run our strategies automatically. After the implementation stage, we run tests using Virtual Machine (VM)s to simulate IoT devices. The tests consist of different numbers of VMs. We perform the test runs with and without VMs simulating crashes. Then comes the evaluation phase, where we evaluate the three strategies (ACO, Random, and Gossips). Figure 1.1 shows the different stages of the master thesis.

The master thesis contributes to the field of decentralized task coordination in heterogeneous and dynamic IoT clusters by addressing the challenges of designing a distributed ACO-based algorithm for resource-constrained devices aiming to reduce the number of messages sent while still guaranteeing that offloaded tasks are processed. The contribution of this thesis lies in the following areas:

1. **Design and development of a decentralized task coordination strategy.** The proposed strategy for task coordination in a decentralized IoT cluster involves ACO-based, Gossip-based, and Random algorithms.
2. **Evaluation of decentralized algorithms.** We evaluate and compare the performance of the strategies through experiments. The result provides insights into these different algorithms' strengths and weaknesses and enables further development of more efficient and reliable task coordination strategies.

Overall, the thesis will contribute to the advancement of the field of IoT by providing a deeper understanding of how decentralized algorithms can be used to coordinate tasks in a dynamic IoT environment efficiently.

We want to emphasize here that we work closely with our colleague, Geri Zeqo. Task coordination between heterogeneous devices can only happen with communication. Zeqo's thesis is responsible for the communication part. Working together, we developed the framework and did the testing. We focus on task coordination in this thesis. To learn more about communication, we refer to Zeqo's master thesis [23]. The two theses combined give a unified and more complete picture of how IoT devices can work together in clusters in a decentralized and autonomous manner – a paradigm shift from the cloud-centric IoT model.

The remainder of this thesis is organized as follows: In Chapter 2, We explain the main concepts for this work, including IoT clusters and swarms, how ACO works, and the difference to the Gossips-based approach. We review related work and analyze if similar

---

works exist in Chapter 3. Chapter 4 describes the methodology of this thesis with the tech stack, design choices, and the system’s architecture. Chapter 5 focuses on the experiments and the evaluation of the three algorithms. In Chapter 6, we write about the discussion of the work. We give context to the reader about potential use cases and the identified limitations of our thesis. Finally, we conclude this thesis with Chapter 7, revisiting our observations and looking at possible future work.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar  
The approved original version of this thesis is available in print at TU Wien Bibliothek.

# Background

This chapter describes the three main concepts on which the master thesis is built. Here we describe why the current central IoT model, which is entirely dependent on the cloud, is not future-proof. The literature recommends a decentralized approach in which IoT devices can gather in clusters or swarms and share resources among themselves. This concept gave rise to the idea of this master thesis, which attempts to solve one of these new challenges, namely the decentralized coordination of tasks. It tries to find completely decentralized non-overloaded devices in a cluster that can assist, considering the nodes' resources and the current cluster state.

We also give a detailed explanation of the two technical concepts ACO and Gossip Protocol, which count as the heart of task coordination in our framework. Our probabilistic task coordination algorithm is strongly based on ACO and tries to compete against the deterministic<sup>1</sup> but much more resource-intensive Gossips-based algorithm.

Section 2.1 describes the paradigm shift from the centralized IoT model to autonomous IoT clusters, followed by Section 2.2, which presents the ACO algorithm in more detail. Last but not least, we explain the Gossip protocol in Section 2.3.

## 2.1 The New Decentralized IoT Model

IoT devices are expanding into more and more critical areas of our lives. However, the majority of these devices are resource-constrained, leading to several shortcomings, such as insufficient computing power for further data processing or limited onboard storage capabilities. Therefore, the data and control of IoT systems are offloaded to cloud facilities to minimize these drawbacks [19]. However, this centralized cloud-centric

---

<sup>1</sup>When we say deterministic, we do not mean that the same result always appears, but Gossips exhibits deterministic behavior in the search space. We will use the term deterministic in this context throughout this paper.

architecture does not scale for complex IoT systems and their requirements. The need for low-latency communication, especially used in autonomous driving, brought the edge and fog computing model [9]. It supports the interaction between IoT and cloud computing to reduce communication delays and the overall data size required to be transferred across the internet and data centers [20].

However, the previous assumptions of the cloud-centric model and the expansion of Edge need to be revisited because IoT systems are growing in complexity and variety. Sensors, actuators, robots, drones, cars, and routers pose new cloud and data center scalability requirements. Additionally, on the one hand, there is a significant increase in resource capacity and capability at the network edge [19]. On the other hand, new IoT devices with significant computing and storage power are emerging. These machines are not constrained to specific data centers but are located in an open space and can act as resource providers themselves [9]. The current centralized model does not consider these new requirements and offers little support for the traditional limitations of IoT devices, such as disruption at any time, heterogeneity, deployment in adverse environments, etc. [19]. The central system architecture is not only struggling with new challenges but also shows itself as a bottleneck for existing problems: Many application use cases are limited by the current cloud-centric programming model because the transfer of large amounts of data to data centers is not sufficiently fast and reliable [20]. The ever faster-growing data volumes lead to inefficient communication bandwidth usage and energy consumption. Moreover, privacy concerns and legal issues arise when data processing and storage happens somewhere other than the IoT devices: IoT devices and the hosted software components may belong in different administrative domains and legal jurisdictions [19]. All these existing and new functional and non-functional requirements imply that computation and control of data must happen closer to the IoT devices. While the cloud can strongly increase IoT applications' computation and storage availability [20], the new paradigm must extend the concepts of the central cloud environments. The decentralized approach should consider: IoT software components face continuous disruptions. Internal faults may decrease the service availability, and the connection to cloud centers is unstable. The current device's circumstances may be untrusted – inside heterogeneous IoT ecosystems, consisting of many nodes and scales, from sensor networks to software components. Finally, IoT devices are, in most cases, resource-constrained, low-powered, and often deployed in unknown or adverse environments [19].

The abovementioned problems lead to a re-orientation of the cloud-centric IoT model towards a more adaptable and decentralized IoT programming model to meet the demands of the emerging IoT use cases and to build more resilient IoT systems [20]. An on-demand, opportunistic, ad-hoc approach is needed that dynamically manages the resource management and architecture without the tight coupling to the cloud [9]. This method targets highly decentralized and federated settings where various providers offer resources and support for IoT devices. This environment stretches from IoT devices to edge nodes to cloud facilities. Resources move dynamically and autonomously between this continuum, depending on where the workload is heaviest. Villari et al. refer

to the shifting of resources along the continuum as Osmotic Computing. The term originates from the movement of solvent molecules through a semipermeable membrane into a region of higher solute concentration to equalize the solute concentrations on the two sides of the membrane [20]. An additional decentralized approach is that the dynamic resource distribution happens within so-called swarms. An IoT swarm is a collection of different nodes from the IoT, edge, and cloud domain that can be seen as a distributed, opportunistic, collaborative, heterogeneous, self-managed, sensing, and learning environment. Swarms are constructed on-the-fly: Devices can join and leave these swarms at any time. Within a swarm, nodes collaborate to accomplish their needs and depend on dynamic, diverse, and distributed resources. Thus, all available heterogeneous resources are used to satisfy specific current demands. From a technical point of view, a swarm is a peer-to-peer system consisting of autonomous devices that sense, compute and make decisions in a coordinated manner to solve tasks. The swarm aggregates resources and collectively complements local knowledge thanks to the connections between the individual devices. The coordination of a swarm or cluster happens in an open and distributed self-managed runtime mode to manage the volatility and uncertainty of real-world IoT use cases better[9]. We use the terms "swarm" and "cluster" interchangeably.

This new paradigm shift needs novel techniques spanning IoT systems' design and management, which introduces new challenges when building such distributed IoT clusters. Questions arise regarding configuration and networking: What might a network abstraction look like that spans from cloud to edge to IoT devices to improve communication performance? There are security concerns to prevent injection from malicious nodes and DDoS attacks. Besides, IoT devices transmit sensitive sensing data that needs to be encrypted efficiently and quickly. Another primary unresolved concern is decentralized orchestration and collaboration among system participants. The runtime orchestration of resources is a complex research topic because it is very challenging to estimate the workload regarding data volume, data arrival time, processing time, etc. [20]. The coordination between the devices to meet the application requirements plays a crucial role in developing dynamic adaptive systems [19]. It must resist short- and long-term network and device failures by self-learning and quickly adapting to new circumstances [9]. In this master thesis, we consider the topic of decentral task coordination in more detail. We ask how we can solve the tasks of IoT devices in a swarm so that the devices support each other collaboratively, even though they are heterogeneous and prone to failures. Our first research question emerged from this motivation: How to efficiently coordinate tasks in a dynamic, resource-constrained, and decentralized IoT setting, where nodes can fail, leave, and rejoin the network at any time? The task coordination strategy must operate without the a-priori knowledge of available resources because we cannot assume a static infrastructure or a central coordination point. Also, the strategy must be able to adapt over time and learn how IoT devices can handle the load. Finally, the strategy must refrain from introducing heavy overhead to the devices and the network. I.e., the number and size of messages must be minimized while keeping the runtime of the strategy low. Chapter 4 provides more information about the requirements and design decisions for the task coordination strategy.

## 2.2 Ant Colony Optimization

Ant Colony Optimization (ACO) is a class of probabilistic optimization algorithms based on ants' foraging behavior, using pheromones to communicate and coordinate food search. Margo Dorigo introduced this swarm intelligence method in his Ph.D. thesis in 1992 [4]. Since his first approach, aiming to find the shortest path in a graph, this optimization technique has solved many numerical problems. Further modifications and extensions of his work have emerged that consider new aspects and limitations from the first attempt.

ACO is based on the behavior of ants, which are considered rather primitive individuals. A characteristic of these animals is decentralized decision-making based on feedback and random behavior. Ants communicate with each other using pheromones, a secretion that they release in their environment. With the help of these capabilities, ants can be seen as a self-organizing system capable of solving complex problems such as building geometric mounds or finding the best path to the food source. When foraging, these animals wander randomly through the area. Once they have found food, they return to the place of origin. In doing so, they leave pheromone trails along the way. Subsequent ants are likely to follow the path and reinforce the pheromone trail if they have also successfully found food. This path can then be considered as a solution. Over time, the pheromones evaporate to allow the possibility of new trails but also to show that sparsely visited trails are less attractive. In ACO, the artificial ants show a few differences from the natural ones: for example, the artificial ants move only in one direction and use defined rules for deciding which edge they will take after each node in the graph. Also, pheromones are dropped after the end of an iteration. Moreover, while Dorigo et al. optimize the problem by the shortest path, natural ants also consider other environmental aspects [6].

The pseudo-code in Algorithm 2.1 presents the simplified idea of the initial algorithm, Ant System (AS), of ACO [6]. It is assumed that  $m$  ants will traverse a graph in a total of  $T_{max}$  iterations. Each ant starts at the same initial node. From there, the state transition rules decide with what probability  $p$  the  $k^{th}$  ant will move from node  $i$  to node  $j$ . In the state transition rules, the following two pieces of information are considered:

1. **a-priori or local knowledge.** I.e., the desirability of a state transition is the length of the path.
2. **a-posteriori or global knowledge.** I.e., the amount of pheromones deposited.

Once all ants have reached the target node, one iteration is over, and the pheromone trails are deposited on each edge, depending on the quality of the path found. Furthermore, a certain amount of pheromones evaporates after each iteration to avoid too fast convergence and thus local optimum. The state transition rules and updating pheromones are described again in more detail in Chapter 4.

---

**Algorithm 2.1:** AS in pseudo-code.

---

**Input:** Number of ants  $m$ , maximum number of iterations  $T_{max}$

**Output:** Best solution found

```

1 Initialize pheromone trails on all edges
2 for iteration  $t = 1$  to  $T_{max}$  do
3   for ant  $k = 1$  to  $m$  do
4     node  $i \leftarrow$  start node
5     while  $i$  not target node do
6       Choose next node by applying the state transition rule
7     end
8   end
9   Compute path quality for all ants and store best path
10  Update pheromone trails on edges by applying the pheromone update rule
11 end

```

---

In general, ACO is applied to solve heuristic NP-hard optimization problems. The application areas of the various ACO algorithms are in the field of combinatorial optimization problems for graphs. Among them belong scheduling problems, vehicle routing problems, multi-depot vehicle routing problems, periodic vehicle routing problems, and split delivery problems. A well-known real-world use case is AntNet, which tries to optimize routing data packets in the network considering the current load [3].

The advantages of ACO are that the algorithms can be adapted to different tasks and find reasonable solutions relatively quickly. Due to the decentralized approach of the ants, tasks can be easily parallelized to manage large-scale problems. Moreover, ACO can be adapted to perform well on dynamic problems where the graph structure changes by adding and deleting nodes and edges. However, ACO has some disadvantages, including relatively slow convergence and high parameter setting sensitivity. Furthermore, ACO is only suitable for use cases on graphs. I.e., the problem needs to be formulated as a graph problem.

ACO sounds like a promising approach for task coordination of decentralized IoT devices because of its already decentralized nature, which can meet the requirements of such systems. Decentralized task coordination in a network of IoT devices is an NP-hard problem because the complexity increases exponentially as the number of IoT devices increases. The main advantages of using ACO in distributed IoT systems are its ability to solve combinatorial optimization problems efficiently. Additionally, ACO algorithms can be adapted to incorporate problem-specific knowledge or constraints, making them suitable for a wide range of IoT applications. ACO has been applied to diverse IoT-

related use cases, including routing in wireless sensor networks [8], and task offloading in fog computing [13]. Chapter 3 describes further use cases in more detail. However, applying ACO in decentralized IoT systems also has disadvantages and challenges. One challenge is the increased communication and processing overhead because devices need to calculate, exchange, and update pheromone information. It can be particularly challenging for resource-constrained IoT nodes, where processing power, energy consumption, and bandwidth/latency limitations are essential concerns. Finally, the performance of ACO algorithms is often sensitive to parameter settings, which may require careful tuning or the development of adaptive strategies to ensure good performance in diverse IoT scenarios.

### 2.3 Gossip Protocol

The Gossip protocol is widely used in communication networks of distributed systems. This protocol, also called an epidemic protocol, is the process of spreading information in a peer-to-peer network to all nodes. It is often used in those application scenarios where the usage of a central entity is not applicable for disseminating data to all network participants. The name and the idea originate from the social aspect of spreading rumors or epidemics. We use the terms "Gossip protocol", "epidemic protocol", and "Gossips" interchangeably.

The Gossip protocol's underlying concept is comparable to the analogy of spreading rumors in school. We assume a student has heard exciting information on the way to school. The child spreads this information to his neighbor, who in turn tells the story to his friend in the schoolyard. The process roughly doubles the number of people who have heard the rumor. The spread of knowledge in networks works similarly in the Gossip protocol: each node in the network periodically exchanges information with a subset of nodes. This subset is usually the set of immediate neighbors of each node. The goal of the protocol is for each node in the network to receive global information. Even though each node has only a local view of the network. This global information exchange often happens through several periodic updates of the nodes. There are several variations of gossip protocols, including push-based and pull-based strategies, each with benefits and trade-offs regarding message complexity and convergence speed.

Gossip protocols hold significant potential for decentralized IoT systems since they provide a resilient and deterministic solution for information dissemination in dynamic networks of heterogeneous devices. In the context of IoT, the advantages of Gossip protocols include the ability to handle dynamic environments, such as frequent node failures and changes. In IoT swarms, Gossip protocols can be used to find required resources without global environmental knowledge. Other assisting tasks include data aggregation, distributed computation, and network monitoring. Chapter 4 explains how we use a naive Gossip protocol implementation to retrieve potential candidates for offloading tasks. The process consists of a discovery phase, where gossip messages are spread across the cluster to find IoT nodes that can assist with a specific task. Afterward,

the overloaded node contacts another node in the second phase based on the discovery phase output.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar  
The approved original version of this thesis is available in print at TU Wien Bibliothek.

## Related Work

In this chapter, we summarize related work, which serves as the basis of our thesis. We review aspects of the decentralized IoT model and highlight work conducted by other authors who have worked with ACO in the context of distributed systems and IoT.

The remainder of the chapter is organized into three parts: of Section 3.1 describes the paradigm shift to the decentralized IoT model. Section 3.2 discusses another master thesis in close cooperation that deals with semantic communication. Last but not least, Section 3.3 summarizes approaches of ACO in distributed environments.

### 3.1 Decentralized IoT Model

The paper *Towards a Cognitive Compute Continuum: An Architecture for Ad-Hoc Self-Managed Swarms* by Ferrer et al. [9] introduces a vision for so-called Cognitive Computing Continuum (CCC), a distributed, opportunistic, and self-managed collaboration concept between heterogeneous devices outside traditional data center boundaries. The authors present an architecture for implementing dynamic resource management in CCCs, aiming to bridge the gap between edge and cloud environments while maintaining Quality of Service (QoS) parameters.

The paper envisions CCCs as distributed, opportunistic, collaborative, heterogeneous, self-managed sensing and learning devices forming so-called swarms. The focal point of CCCs is cognitive devices, which autonomously make decisions using onboard computation and storage capacities based on sensed environmental information. Unlike traditional distributed computing concepts, these devices exchange acquired status information and make critical decisions for the system's correct functioning.

The proposed architecture relies on an extended, context-aware overlay P2P-based network controlled by middleware, which bootstraps swarm building and coordinates activities to provide the required QoS despite expected node fluctuations and service disruptions. The authors outline the central CCC characteristics and introduce their three

fundamental properties: awareness, autonomy, and actionability. The paper provides an initial approach to realizing the vision of CCC. The concept can be seen as a highly decentralized IoT system.

*Osmotic Computing: A New Paradigm for Edge/Cloud Integration* by Villari et al. presents a computing paradigm that addresses the limitations of existing cloud-centric IoT programming models [20]. The paradigm aims to improve the dynamic management of services and microservices across cloud and edge data centers, ensuring QoSs for IoT applications.

The motivation behind osmotic computing lies in the emerging complexity of IoT devices and the large volumes of data they generate, which can lead to bottlenecks in traditional cloud computing systems. The authors propose a more adaptable and decentralized model that utilizes resources from both cloud and edge infrastructures in a federated environment.

The term osmotic computing originates from the concept of osmosis in chemistry, which refers to the seamless diffusion of molecules from a higher to a lower concentration solution. This process should represent how services can be migrated across data centers to the network edge. In osmotic computing, the dynamic management of resources in cloud and edge data centers evolves toward the balanced deployment of microservices, similar to how molecules in a solution move across a semipermeable membrane to achieve equilibrium. However, unlike the chemical osmotic process, osmotic computing allows for a tunable configuration based on resource availability and application requirements, determining whether microservices should migrate from cloud to edge or vice versa. The paradigm extends the traditional notion of runtime control and reconfiguration to resources deployed and available at the edge. It also exploits lightweight container-based virtualization technologies, such as Docker and Kubernetes, for deploying microservices in heterogeneous edge and cloud data centers.

The authors highlight the need for machine learning techniques to develop predictive models for forecasting workload input and performance metrics across multiple microservices on cloud and edge data center resources. They also emphasize the importance of intelligent, QoS-aware, and contention-aware resource orchestration algorithms based on these models, monitoring systems, and configuration selection techniques.

In the paper *Towards Resilient Internet of Things: Vision, Challenges, and Research Roadmap* by Tsigkanos et al. [19], the authors argue that resilience is a crucial property of IoT systems due to their complexity, dynamic nature, and increasing penetration into critical aspects of human activity. They call for a paradigm shift in the design and operation of IoT systems, moving from centralized cloud-IoT configurations to decentralized systems. It would involve novel methods spanning the design and operation of IoT systems.

Resilient IoT systems should be able to fulfill requirements in changing, unpredictable, and potentially adversarial environments without relying on a central control point. The authors propose a research roadmap to address these challenges, focusing on characterizing resilience, analyzing IoT systems for maintaining resilience in the absence of central

control, and investigating the role of data in such system. They also emphasize the need for monitoring, validation, and countermeasures at runtime to maintain resilience.

By leveraging distributed systems and formal software engineering research techniques, the authors suggest that decentralized coordination, control, and data management will be vital to achieving resilience in IoT systems. They also advocate for treating IoT systems as self-adaptive systems, continuously monitoring and reacting to changes in their environment to maintain resilience objectives. The paper outlines a vision and research roadmap for building resilient IoT systems that can handle disruption and change in the face of increasing complexity and integration into critical aspects of human life.

The authors of all three papers envision the future of IoT in a decentralized architecture. The systems must move from the cloud-centric narrative to a dynamic, self-adapting, self-learning, and autonomous structure. This paradigm shift has given rise to the origin of our work. We examine the subject of task coordination in this centralized context.

### 3.2 Adjacent Thesis

As read in the last section, the evolution of IoT systems is transitioning from a centralized to a decentralized approach. The vision imagines decentralized autonomous clusters of IoT devices that can act independently. However, to advance in this direction, many issues and challenges still need to be solved in the large topic of decentralized IoT clusters. One of these is task coordination, which we address in this work. Another critical point is the communication between devices in the cluster. A functioning system can only be established on a coordination strategy, which in turn depends on communication mechanisms in order for task exchange to happen at all.

For this reason, Zeqo tackles the topic of decentralized communication or, more precisely, semantic communication [23] in his work *Decentralized Communication in Heterogeneous IoT Clusters*. We describe aspects of this vital topic in Chapter 4. Our two separate works depend on each other, so our two master theses were done in cooperation. While task coordination is covered in this work, semantic communication is described in Zeqo's thesis. For this reason, we focus on different metrics in the jointly conducted experiments. Our work can be seen as puzzle pieces, providing a clearer picture of how decentralized IoT systems might operate. They serve as a foundation for further work in that direction to find the other missing puzzle pieces.

### 3.3 ACO-related Work

The following paper by Dorigo and Gambardella introduces the Ant Colony System (ACS), a distributed algorithm applied to the Traveling Salesman Problem (TSP) [5]. The algorithm is inspired by the behavior of real ants, which use pheromones to communicate indirectly and find the shortest path between food sources and their nest. ACS uses a set of cooperating agents called ants to find suitable solutions by depositing pheromones on the edges of the graph.

ACS improves on the previous AS [6] by having ants search in parallel for reasonable solutions and cooperate through pheromone-mediated indirect and global communication. Each ant constructs a solution candidate iteratively, using information from past experience and a greedy heuristic based on edge length. The paper studies the proposed algorithm by examining its characteristics, such as pheromone changes during runtime, estimating the optimal number of ants, and evaluating the role of pheromone and the greedy heuristic. Results show that their work outperforms other nature-inspired algorithms like simulated annealing and evolutionary computation. The ACS augmented with a local search procedure, called ACS-3-opt, is compared favorably with some of the best-performing algorithms for symmetric and asymmetric TSPs.

The authors suggest that the ant colony approach can be improved by changing the number of ants contributing to global updating, moving from parallel local updating of pheromones to a sequential one, and adding a more effective local optimizer. Ongoing research aims to establish the class of problems that can be tackled by ACS, with preliminary results showing potential in combinatorial optimization problems like the quadratic assignment problem. In our work, we use the state transition rules formulas and evaporation concepts like in the AS and this paper. However, we encode it for our problem domain and adapt it to fit decentralized problems better where traversing the graph is expensive.

The paper *Ant Algorithms for Search in Unstructured Peer-to-Peer Networks* by Michlmayr presents SemAnt, an algorithm for distributed query routing in peer-to-peer networks inspired by the ACO meta-heuristic [16]. The research aims to determine the feasibility of using ant algorithms for content-based query routing in peer-to-peer networks, considering their inherent dynamics, such as frequent content repository changes and peers joining or leaving the network.

SemAnt is designed for a distributed search engine where each peer manages a repository of documents and offers its content to other peers while performing keyword-based searches based on metadata.

The algorithm employs forward and backward ants to support distributed problem-solving and prevent cycles. Forward ants are created for each query and have a Time-To-Live (TTL) parameter to prevent infinite runs. If a forward ant finds a peer with documents satisfying the query, it creates a backward ant. The algorithm's performance is measured by resource usage (number of links traveled for each query) and hit rate (number of documents found for each query). SemAnt's performance is evaluated by simulating a peer-to-peer network with 1,024 peers and comparing it to the well-known k-random walker approach. The results show that SemAnt converges quickly, reaching optimal performance after 2,000-time units. The hit rate increases to 3.95 documents per query, while resource usage decreases to 54.04 links traveled. The hit rate remains nearly constant from time unit 2,000 to 10,000, while the resource usage decreases slightly to 53.02 links per query at time unit 10,000.

Overall, SemAnt demonstrates the potential for using ant algorithms in peer-to-peer networks, offering robust results and fast convergence. Michlmayr's work and her technical report [17] serve as an essential foundation for our thesis. Thanks to the results, we can

conclude that ACO could be a potential candidate for task coordination in distributed environments. Our work differs because we do not have a pure peer-to-peer environment and try to outsource tasks instead of investigating whether a given node owns a given resource. Nevertheless, we could reuse valuable aspects of this work and adapt them to our objectives.

The next paper presents SoSACO-v2 [2], a bio-inspired algorithm that extends ACO by giving ants a sense of smell, which helps them find paths between nodes in large dynamic graphs quickly. The primary motivation behind this extension is to address challenges faced by traditional path search algorithms that do not scale well and are unsuitable for dynamic environments. SoSACO-v2 is designed for domains with optional optimality, and the path search takes longer than covering the path.

SoSACO-v2 improves upon its predecessor by providing fast pathfinding between any two nodes within a huge graph and refining the solution if extra time is available. It is achieved by utilizing divide and conquer techniques, using advantages supported by the food node, and including a variant of the tabu list extension. The algorithm may not always provide the optimal path but responds significantly faster than other algorithms. The paper evaluates SoSACO-v2 on a sizeable generic graph and a small-world type graph from a real social network, showing satisfactory results. The proposed algorithm can also refine its solution when given more time, making it suitable for applications that require quick responses and improvements within a specific time limit. Future research aims to enhance the algorithm's performance in dynamic graph scenarios and improve its adaptability in changing environments.

Although exciting concepts are brought forth in this paper for graphs of enormous size, we do not consider this paper in our thesis because we assume that our IoT systems do not consist of so many nodes – for the sole reason that we cannot conduct performance tests for such an amount of VMs due to lack of resources.

IoT environments generate a large amount of data, making the routing and scheduling of data from source to destination challenging. In the paper *Comparative Study on Ant Colony Optimization (ACO) and K-Means Clustering Approaches for Jobs Scheduling and Energy Optimization Model in Internet of Things (IoT)* by Kumar et al. [14], the authors propose a comparison between ACO and K-Means clustering algorithms for IoT job scheduling. These two approaches are studied to find the shortest route path, optimize QoS constraints, and reduce energy consumption. The authors suggest dividing the IoT environment into areas and clusters based on network types.

The primary objective is to calculate and compare the response time for message forwarding using ACO and K-Means clustering algorithms, ultimately reducing energy consumption. The performance of both algorithms is analyzed using parameters such as average queue length, waiting time, and response time. The results show that ACO provides better performance for the considered parameters.

The paper concludes that efficient message forwarding in IoT environments using ACO can optimize energy use and ensure a more effective IoT system. This work contributes to understanding the potential benefits of ACO and K-Means clustering algorithms for

job scheduling in IoT. It provides a basis for further exploration and development.

Compared to our work, the authors consider service rate, waiting time, and queue length metrics but do not address whether ACO scales as the number of nodes grows. They do not discuss the fact that a high number of ants and iterations clutter the network with messages. In addition, no tests were performed in a dynamic environment with device failures.

Task allocation is a crucial aspect of agent cooperation in Multi-Agent Systems (MAS), and challenges such as network latency, dynamic topology, and node heterogeneity demand innovative approaches. Wang et al. present the Collection Path Ant Colony Optimization (CPACO) method, which improves upon traditional parallel computing task allocation methods and ACO algorithms [21]. The work addresses ACO's issues by modifying the heuristic function, updating the strategy in the Ant-Cycle Model, and creating a three-dimensional path pheromone storage space. The paper details the application of CPACO to multi-agent task allocation, comparing its performance with the Global Search Algorithm (GSA) and Forward Optimal Heuristic Algorithm (FOHA) methods. Experimental results reveal that CPACO consumes only 10.3% of the time GSA takes while outperforming FOHA. CPACO is advantageous when dealing with large-scale MAS task allocation, offering rapid convergence to optimal solutions. However, it may fall into local optima and can be sensitive to initial parameter selection.

The study concludes that CPACO has significant potential for optimizing task allocation in MAS, with plans to refine its optimization performance and explore additional applications to meet diverse system requirements. The paper differs significantly from our work in the experimental environment and thereby has implications for the design decisions of the algorithm. The authors simulate the test environment with a grid while we use VM instances and are thus closer to real-world use cases. The paper does not focus on IoT systems but on general MASs. Therefore, aspects such as limited resources, expensive communication, etc., need to be addressed.

The paper *Efficient Task Offloading for IoT-Based Applications in Fog Computing Using Ant Colony Optimization* by Hussein and Mousa focuses on improving QoS for IoT applications using fog computing instead of cloud computing [11]. Fog computing enables lower latency for delay-sensitive applications compared to cloud computing. The authors propose two nature-inspired meta-heuristic schedulers: ACO and Particle Swarm Optimization (PSO), for load-balancing IoT tasks on fog nodes while considering communication cost and response time.

A formal IoT-fog system model is presented, aiming to minimize task response times while considering network bandwidth, latency, and existing load on fog devices. Experimental results demonstrate that the proposed ACO-based scheduler outperforms the PSO-based scheduler and the round-robin algorithm regarding response time and effective use of fog nodes.

Future work could include extending the research in two ways: (1) exploring multi-objective optimization, considering power consumption, communication cost, computation cost, and task dependencies; and (2) investigating dynamic scenarios involving non-

stationary IoT sensor nodes, mobility, and adaptive sampling techniques that produce dynamic data rates. In contrast to our thesis, they use a centralized architecture in which requests are forwarded to a master fog node, which is then responsible for scheduling.

The paper *A Task Allocation In IoT Using Ant Colony Optimization* by Zannou et al. addresses the issue of task allocation in IoT by proposing an algorithm based on ACO [22]. The authors propose a strategy to distribute task capabilities to the most competent nodes for reducing resource consumption. Their approach considers two parameters: the optimal path length and the capabilities of the nodes. ACO is then applied to find the best path nodes and distribute the task allocation capabilities accordingly.

The authors categorize capabilities into three types: fixed, maximal, and minimal, and adapt the ACO algorithm to take into account the location and capabilities of the nodes. They run simulations using data from the Stanford Large Network Dataset Collection, comparing their strategies' performance with a standard scenario without applying any algorithm.

The results show that the proposed algorithm is efficient in finding the shortest path distance for task execution and obtaining efficient capabilities corresponding to the desired capabilities. The authors suggest future work to study the similarity of task capabilities to predict the execution path.

Similar to our work, we encode the path length and capacities of the nodes in the pheromones. However, this paper does not consider the aspect of the dynamic nature of IoT devices in the experiments and whether such an approach with multiple iterations is realistic for a decentralized IoT system.

In the paper *Task Offloading in Fog Computing for Using Smart Ant Colony Optimization* by Kishor and Chakarbarty, the authors address the latency issue in cloud computing for time-sensitive real-time IoT-sensor applications [12]. They propose a meta-heuristic scheduler called Smart Ant Colony Optimization (SACO) to offload tasks in a fog environment. The paper focuses on the comparison of this algorithm with existing approaches like round-robin, throttled scheduler algorithm, modified PSO, and Bee life algorithm.

The paper demonstrates that fog computing can fulfill the low-latency requirement of QoS in IoT-sensor applications, making it more suitable for handling large volumes of multimedia data than cloud computing. The proposed SACO strategy significantly improves latency when offloading IoT-sensor application tasks. It reduces task offloading time by 12.88%, 6.98%, 5.91%, and 3.53% compared to RR, throttled, modified PSO, and Bee life algorithm, respectively. In future work, the authors plan to extend the research to include network latency and power consumption reduction and potential applications in military, smart agriculture, and weather forecasting. Additionally, they intend to explore the effects of mobility in sensor nodes.

However, the work uses a three-layered infrastructure consisting of sensor devices, fog nodes, and the cloud center. This design choice introduces a centralized component, namely the fog layer, while we focus on a highly decentralized setting where each node is considered autonomous.

### 3. RELATED WORK

---

Bui and Jung's study explores the potential of ACO for dynamic decision-making in connected vehicles within an IoT environment [1]. The researchers propose a model for the next generation of intelligent transportation systems, in which vehicles share information and autonomously adapt to find optimal paths to their destinations. To do this, they first establish a communication framework for connected vehicles to share traffic flow information. Then, they apply the concept of Swarm Intelligence (SI), treating connected vehicles as artificial ants that self-calculate to make adaptive decisions based on traffic flow dynamics.

The study focuses on several research questions related to how connected vehicles communicate and collaborate in an IoT environment, the information needed for applying ACO among connected vehicles. They simulate various transportation scenarios to evaluate the effectiveness of their ACO-based approach for connected vehicles, comparing it to previous methods.

The simulated results show promising improvements in vehicle waiting times, especially in scenarios with high traffic density. The authors conclude that their approach is worth further exploration. They plan to apply it to different ITS scenarios and investigate hybrid SI-based algorithms for IoT systems in the future.

# Methodology

This chapter presents the technical aspects of this master thesis. The goal is to give the reader a concise understanding of the unified framework's technical components. We answer in this chapter the first research question of how decentralized task coordination can be achieved in a heterogeneous and dynamic IoT cluster.

The following Section 4.1 forms the foundation of the framework. It outlines the requirements and goals the implemented prototype should consider and accomplish. We describe in Section 4.2 the system architecture of the framework with the individual software components. While Section 4.3 explains the three strategies, Random, ACO, and Gossips, for decentralized task coordination, we briefly look at swarm communication in Section 4.4. Finally, Section 4.5 illustrates how a monitoring component is responsible for collecting performance data. This data is used for the evaluation of the strategies.

## 4.1 Requirements & Objectives

The starting point of this work is the reconsideration of the cloud-coupled-centric IoT model into a more decentralized flexible paradigm. Chapter 2 describes the motivations and emerging challenges with the introduction of this shift. Flexible and distributed coordination that better considers the available resources, heterogeneity, and dynamic nature of IoT devices is demanded. This thesis examines decentralized task coordination in distributed IoT systems and implements a solution approach. We hope that the output of this master's thesis provides a foundation for further research.

Starting from the topic of task coordination, we have done requirements engineering to draw the exact boundaries of our work. This phase served us to define the objectives and scope. Based on that, we could decide which methodologies to apply for the thesis: Requirements Engineering as the definition for the content of the work, state-of-the-art

research as the decision maker in which direction to go further, prototype implementation as proof of concept for our work, and experiments for evaluating our strategies.

At the initial stage of the requirements engineering phase, we constructed a use case as the basis for our requirements and decisions: Assume that there are a certain number of IoT devices. They are equipped with different processing, storage, and communication capabilities. Each IoT node has one or more emitters that periodically produce tasks. Due to the limited queue size, a device may be overloaded over time and need support to keep up with processing all the tasks. For this reason, attempts are made to offload tasks to other devices in the cluster. Because these devices are connected to each other, they form a cluster of different nodes that can help process the task. The master thesis aims to implement coordination between devices to offload tasks.

This coordination must consider the following challenges of distributed IoT systems:

- **Heterogeneity.** Members of an IoT swarm have different hardware specifications, queue sizes, and emitter types. Different IoT devices support different tasks, and the processing time of a task varies depending on the device's capabilities. Also, some devices are more error-prone than others, which affects the reliability of successful task processing.
- **Processing and communication overhead.** Because most IoT devices are resource-constrained, task coordination must be designed so that it neither negatively impacts the performance of the devices nor overflows the network with messages.
- **Scalability.** There is no upper limit to the size of distributed IoT devices in a cluster. Successful task coordination algorithms must scale with the number of devices and avoid becoming a bottleneck.
- **Dynamic environment.** Task coordination must assume that the setting is dynamic due to the failure and joining of IoT devices at any given time. For this reason, coordination must not assume a-priori knowledge but must be able to adapt to topology changes during runtime. Furthermore, coordination must learn the cluster's topology and available resources during runtime. I.e., the topology must not assume global knowledge.

Based on the above prerequisites, we define the scope as implementing and evaluating a decentralized algorithm for coordinating tasks within an IoT cluster. The focus is firmly on the algorithm. The remaining aspects, like tasks or device properties, are simulated for the purpose of this work and are not part of the thesis. Furthermore, the security aspect needs to be addressed within the context of another work. More on this and other uncovered but essential areas of IoT systems can be found in Chapter 6.

## 4.2 System Architecture

The system's architecture is described in three parts: First, the system is presented in context with the stakeholders and other systems. A description at the level of the running applications follows this. Here we show how the implemented application relates to others. Last, we view our framework at the software component level. In this view, we explain which components exist, what they are responsible for, and how they interact with other software parts.

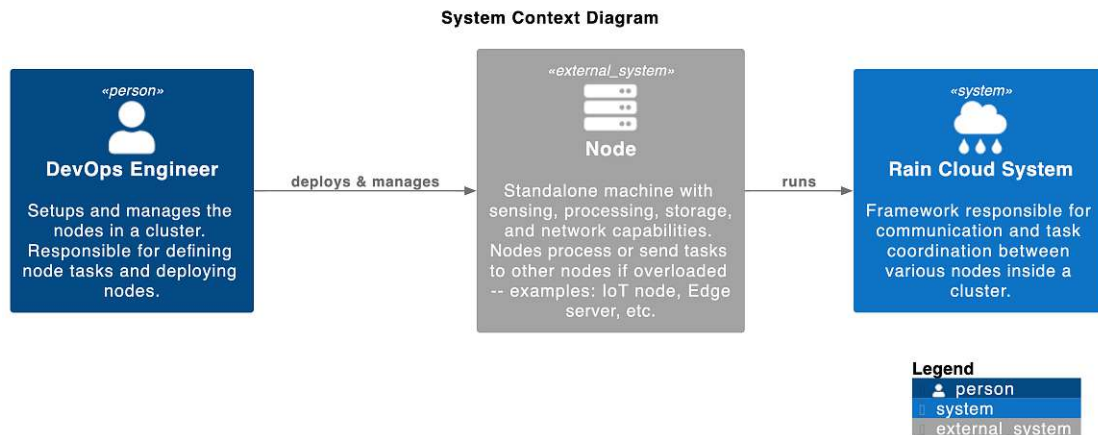


Figure 4.1: System context diagram of the framework.

Figure 4.1 shows the relationship between stakeholders, devices, and the framework. The stakeholder, in most cases a DevOps engineer, is responsible for setting up the IoT devices and the running software that uses the implemented framework. The framework is responsible for enabling machines to connect and communicate with each other. As a result, clusters are created that the DevOps engineer can manage by adding new nodes, removing existing ones, or generally defining which device can connect to whom. It allows the cluster size to be adjusted according to the use case. The framework provides intra-cluster communication and task coordination.

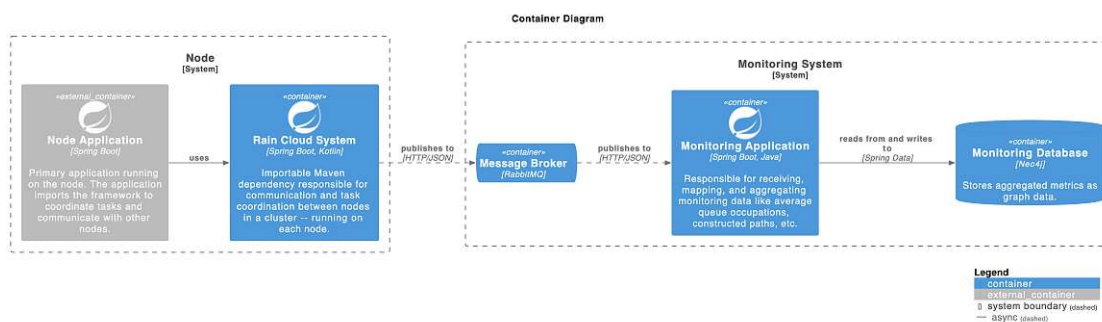


Figure 4.2: Container diagram of the framework.



### 4.2.1 Device Properties

Each device that is located in an IoT cluster formed by the framework has specific hardware resources. This results in devices that can help out other devices, while weaker nodes tend to offload tasks and thus depend on the available resources in the swarm. To simplify the testability of the framework, in this work, we simulate the resources instead of using different VM specifications by mapping them to software-level values. The system has three resource groups (computation, communication, and storage), which can take values between 1 and 5. A lower value means that the device is better equipped. I.e., the strongest devices in terms of hardware are those where all three values are set to 1. The resources directly influence the processing time of the tasks in the system. Devices with lower resource values can process tasks faster.

Each device defines for itself which tasks are emitted and processed internally. Moreover, these tasks are also accepted and thus processed if another node requests that a task be offloaded. If a node wants to offload a task, it appends a TTL in seconds to the request, indicating how long the task is valid during the offloading process. The offload request is ignored if the TTL is over, but no device that can process this offloaded task has been found yet. The idea behind this is that an offloaded task only becomes beneficial for the overloaded device if it is processed within this time. Otherwise, the device can execute the task itself.

Both the device and task properties are summarized as node profiles. When a device connects to other devices in the cluster, the node profiles are exchanged to get more information about the neighbors and new members. The information is essential for task coordination because it indicates what new resources exist in the cluster. As can be seen from Figure 4.3, at the software level, the properties described above are called RCS Properties and Node Profile. Table 4.2 presents an overview of the framework's device and task parameters.

### 4.2.2 Task Coordination Strategies

The strategies are responsible for finding a node from the cluster that provides resources to execute a task because the device from which the task originally comes is overloaded. Figure 4.3 displays three strategies in the coordination module: 1.) Random, 2.) ACO, and 3.) Gossips. During runtime, exactly one strategy is active and controls task coordination. The main difference between the strategies is how they select a candidate for offloading. In the following sections, we give a brief overview, while the operation of each algorithm is considered in more detail in Section 4.3.

In this thesis, the random-based approach, also called Random for short, is left to chance which node is contacted to process the task. Initially, an overloaded node wants to offload a specific task. This machine is located in an IoT cluster and thus has an open connection with other nodes. The device now sends an offload request to one of its  $n$  neighbors. The probability that one of the neighbors is selected is  $1/n$ . The selected neighbor receives the offload request and checks whether it supports this task. If not, the device again

#### 4. METHODOLOGY

Name	Description
computation	Represents the device's processing capabilities. Can be any integer value between 1 and 5.
communication	Represents the device's communication capabilities. Can be any integer value between 1 and 5.
storage	Represents the device's storage capabilities. Can be any integer value between 1 and 5.
supportedTasks	Represents which tasks the device can receive and process when offloaded. Is a list of strings in the format $T_1, T_2, \dots, T_n$ .
functionTimeMultiplyFactor	Represents a constant by which the task processing time is multiplied. Can be any positive decimal number.
timeToLiveFactor	Represents a constant for calculating the maximum duration of the offloading process for a task. Can be any positive integer value.

Table 4.2: Device's and tasks' properties.

forwards it to one of its neighbors using the same selection strategy. If the node knows this type of task, it checks whether it can still process it with its available resources. If not, the offload request is forwarded again to one of its random neighbors. If the next device is not overloaded, it checks whether the TTL is greater than 0. If the TTL has elapsed, then it ignores the request. However, if the TTL is in the positive range, it states there is still time to process the task. Thus the node accepts the offload request with the task attached to it and sends the task's response back on the same path after successful completion unless the return path can be optimized. Chapter 4.3 reveals more about this optimization and the strategy.

The second strategy is also probabilistic and is based on AS, which is why the strategy is called ACO. The offload request, seen as a virtual ant, wanders through the graph carrying the task. As soon as a node is found, the ant hands over the task to the device and sends an ant back with the response of the processed task. ACO's mode of operation differs from Random in selecting the next neighbor. Instead of choosing randomly, the node whose pheromones are the highest is preferred. The pheromones' value is decided by how far the neighbor is from the requestor, how strong its resources are, and how many times a task has been successfully processed. A more detailed description of the selection of the neighbor and the calculation of pheromones is explained in Chapter 4.3.

The final strategy is called Gossips because the search for potential candidates to complete an offloaded task is based on a naive Gossips protocol. This strategy can be classified as deterministic to the search space because, in the first phase, the swarm is flooded with messages to find nodes that can help out. The overloaded machine sends offload requests to all its neighbors, which in turn forward requests to all their neighbors until they hear the request again from another neighbor. If a machine can help, it responds along the

same path. By spreading Gossips messages, one receives multiple candidates to offload. In the second phase of the strategy, the originating node ranks the candidates by resource strength and distance and then contacts the first in the list through the response path to execute the task. The strategy is described in greater depth in Chapter 4.3.

### 4.2.3 Task Queue

Each device that uses the framework has a queue for managing tasks. One or more emitters, acting as producers, periodically place tasks in the double-ended blocking queue. Producers also include devices that need to offload tasks. I.e., those that do not originate from the device itself but from another overloaded machine that attempts to delegate the task to available devices. While there is no upper limit to the queue occupation, each device has a property queue capacity. This positive integer value indicates how many tasks will fit in the queue that the device can execute. Once the threshold is reached, each subsequent task is first attempted to be offloaded before the device processes the task itself. Figure 4.4 illustrates this process.

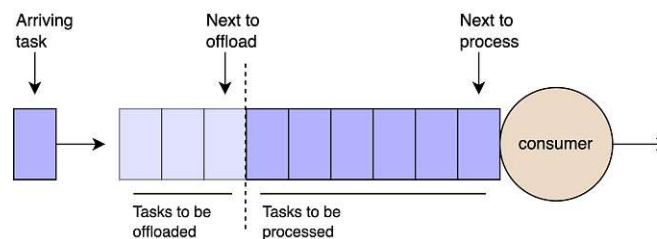


Figure 4.4: Structure of the queue.

Assume the queue capacity is set to 10. I.e., as long as there are less than 11 tasks in the queue at any time, the device will process these tasks. If the arrival rate of the tasks is higher than the processing time of the tasks, more tasks end up in the queue, and the specified threshold is exceeded. Thus the 11<sup>th</sup> task is offloaded. If it is successfully processed by another node in the cluster and the requestor receives a positive response, the task is deleted from the queue since it counts as completed. If there is a lack of available resource in the cluster or a successful response is missing, the task remains in the queue until the device processes it.

The queue shows a partial First In – First Out (FIFO) behavior: The two areas, separated by the queue capacity index, are FIFO individually, but combined, they are not. The idea behind this design decision is that the device can process all tasks within the queue capacity index. Nonetheless, because a later added task can be processed earlier by offloading it, the tasks must be independent of each other – both temporally and logically. Furthermore, every task in the system is stateless and has no input. It means that the task does not have to be added to the offload request, but the node determines what has to be processed based on the task type. We cover more about these constraints in Chapter 6.

### 4.2.4 Task Emitters

In the framework, the emitters are responsible for generating tasks. One can imagine that a sensor emits data, representing the data as a task to process. In order to test the system, test data was used to 1.) simulate the emitter frequency and 2.) processing time of a task. Thus, the emitter not only has the responsibility to emit tasks but defines how long a task takes to process. The emitter reads both the arrival rate of the tasks and the separate task duration from files. Based on the device resources, the read task duration increases by a multiple. The worse the resources, the greater the processing time. More information about the used test data can be found in Section 5.1.2.

### 4.2.5 Communication Module

The right side of Figure 4.3 shows the module responsible for communication between devices in the swarm. While we briefly overview these sections' main components, Section 4.4 describes Node Discovery (ND) and Self-Actualization (SA) in more detail.

Different protocols have been defined for intra-cluster communication, communicating over Transmission Control Protocol (TCP) sockets. The protocols parse incoming and outgoing messages and call the appropriate interfaces. The messages following the JavaScript Object Notation (JSON) standard were kept as compact as possible to avoid overloading the network. The communication module is responsible for two crucial decentralized concepts, ND, and SA.

ND is about how IoT devices can find each other, connect and form a swarm. In the handshake, the node profile of the other device are exchanged. The information includes their device resources and which tasks they support. In the device, it is possible to define how many open connections they should have to other swarm members and on which address they can connect.

SA is a decentralized adaptation strategy for the swarm. Task coordination teaches which devices are reliable over time and how attractive their resources are. SA takes this local knowledge and tries to close connections that count as irrelevant. It opens up the possibility of making new connections and thus obtaining a better swarm topology.

### 4.2.6 Database

The framework employs an in-memory database to maintain the node profiles of the connected neighbors and to delete lost connections. It also stores the pheromone tables that the ACO strategy populates, updates, and retrieves. For the Gossips strategy, it is responsible for saving aggregated statistics, which is used by the SA service.

## 4.3 Task Coordination Strategies

The decentralized task coordination strategies are the primary focus of this master thesis. They are responsible for collaboratively processing tasks in a decentralized IoT system to

use resources better. Section 4.1 mentioned the challenges of the distributed IoT model that task coordination has to address. Based on this, we have developed a strategy called ACO, whose design decisions try to cover the following requirements:

- **Decentralized architecture.** The strategy must be completely decentralized and cannot rely on a central coordination entity. Thus, the strategy must operate with local knowledge instead of global knowledge, which increases complexity.
- **Flexibility.** The strategy must consider the heterogeneity of IoT devices in distributed IoT applications. It includes different processing and storage capabilities.
- **Efficiency.** The strategy must minimize computational and communication overhead because most IoT devices are resource-constrained and cannot apply heavy-weight strategies. On the one hand, the network must not be flooded with messages. On the other hand, the strategy must only work with local knowledge.
- **Scalability.** The strategy must be able to tolerate a high number of IoT devices without experiencing performance degradation.
- **Adaptability.** The strategy must adapt to changes in the network and be resilient to the high mobility of IoT devices. The dynamicity includes allowing IoT devices to leave and enter the cluster at any point in time.

Two more strategies, Random and Gossips, were developed to compare the performance of ACO. We use Random as the baseline algorithm. As the name suggests, task coordination happens here at random. ACO is our self-learning and self-adapting approach that covers the above requirements by trying to remain as lightweight as possible while still being performant. The Gossips strategy is, by design, more accurate at coordination because it has more knowledge about the cluster. However, this knowledge comes at the cost of efficiency.

The following sections describe the three algorithms in detail. The problem of all three strategies can be reduced to a graph instance. The IoT devices are mapped to nodes, while the connections between the devices are represented as edges. We call the overloaded node that wants to offload a task *origin*. The node that can process the task is called *target*. All three strategies aim to find the optimal path between the origin and the target.

### 4.3.1 Random

#### Motivation

The Random strategy is a straightforward technique for decentralized task coordination in distributed IoT systems. Tasks are offloaded to randomly selected nodes without considering their specific characteristics or currently available resources. This approach does not possess self-learning or self-adapting capabilities, limiting its ability to optimize

task distribution and adapt to dynamic changes in the IoT environment. Despite its limitations, the Random strategy offers a straightforward approach to task coordination.

The motivation behind applying the Random strategy lies in its simplicity and ease of implementation, which makes it an ideal candidate for establishing a baseline for performance tests against more sophisticated strategies – in our case, ACO and Gossips. It allows us to measure the potential improvements and benefits of using more advanced strategies in distributed IoT systems. By evaluating the performance of the Random strategy to the performance of ACO and Gossips, we can gain valuable insights into the effectiveness and efficiency of each approach.

### Implementation

A node is called overloaded in the system when the queue reaches a specific size (see Section 4.2.3). The node tries to offload this task by building an offload request. The request contains the task type and a TTL. While the target node uses the former to specify which task it must process, the latter specifies how much time remains for the offload process. The origin node randomly chooses to which neighbor it will send this request. The neighbor receives this message via the respective protocol. If the node cannot accept the request because 1.) the task type is unknown to it or 2.) it is overloaded itself, it forwards the request to another randomly selected neighbor. However, if the two points do not apply, the node checks whether the TTL has expired. If so, the request is ignored, and the task is not offloaded. If not, the device adds the task to the queue. After the origin node has finished processing the task, it sends back a response along the optimized path. How path optimization occurs is explained in Section 4.3.1. In addition to this optimization, a circle detection was implemented in the strategy so that the offload request is not sent unnecessarily in a circle and thus has a negative impact on the latency and bandwidth of the network.

Due to the straightforward structure of this strategy, no pseudo-code is provided. However, the logic can be derived from Algorithm 4.1: The fundamental difference in the random-based approach is that the next neighbor for the offload request is not decided using ACO's state transition rules but at random.

**Path Optimization** Path optimization is performed on the return path when passing the response from the target to the origin node. The purpose is to shorten the entire offload process duration by saving hops to the origin. Figure 4.5 illustrates the process: In the scenario, the dashed lines indicate the node connections. The origin,  $n_1$ , has offloaded a task via  $n_2$ ,  $n_3$  to  $n_4$ , the target device (see numbered lines). Now the target sends a response, with the forward path encoded, back to  $n_3$ . This device, however, has an active connection to the origin and, for this reason, does not forward the response to  $n_2$  but directly to  $n_1$ . The technique saves a total of one hop in this example. In general, it reduces the duration of the offload operation and thus saves messages, relieving the network's latency and bandwidth. The optimization also reduces the probability that the

response cannot be forwarded on the return path due to node failures, which improves reliability in a dynamic IoT setting.

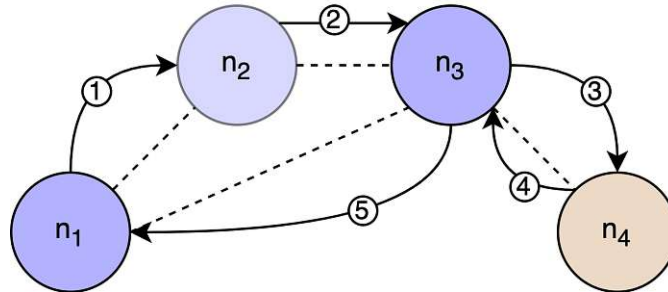


Figure 4.5: Path optimization illustration.

**Time Complexity** The computational complexity of selecting a random neighbor is  $O(n)$ , where  $n$  is the number of neighbors. Thus, the runtime of offloading or forwarding is also  $O(n)$ .

### Characteristics

The Random strategy's architecture entails inherent limitations and potential drawbacks due to its uninformed and non-adaptive nature. One significant limitation of this algorithm is that it does not consider the nodes' load or resource availability when selecting which node to send or forward the offload request. Consequently, it can lead to an improper distribution of tasks across the IoT cluster, potentially overloading some nodes while leaving others underutilized. Furthermore, the Random approach does not consider the network topology. As a result, tasks may be sent to distant nodes, increasing the overall response time and generating more messages within the network. Therefore, the network must cope with increased latency and bandwidth. Finally, the algorithm has not implemented mechanisms for self-learning and self-adapting methods and, for this reason, has a static nature. However, it is deployed in a constantly changing IoT environment. As the network topology evolves and nodes' resource availability changes, the strategy's inability to adjust its decision-making process may lead to further inefficiencies and performance issues.

### 4.3.2 Ant Colony Optimization

#### Motivation

The second decentralized task coordination strategy is based on AS, a member of the ACO family of algorithms, and therefore is called ACO for short in this paper. The most relevant one of the three task coordination algorithms uses artificial ants that traverse the network graph in the IoT system and deposit pheromones between connections. Task coordination uses these pheromones as a decision basis because the quantity indicates

how attractive a path between two nodes is for task offloading. The pheromones are updated during runtime and provide a self-learning and self-adapting character to this approach. However, because the AS algorithm was not designed for distributed IoT applications, our strategy had to be adapted to the requirements of the decentralized IoT model.

The motivation for employing the Ant Colony Optimization (ACO) strategy stems from its adaptability to the problem domain, self-learning capabilities about the system, and ability to react to environmental changes like node failures. These attributes make the ACO algorithm a highly suitable approach for addressing the challenges of coordinating tasks in dynamic and complex IoT systems. We have successfully applied AS to our problem domain by adapting the computation of pheromones to our needs. Furthermore, the artificial ants help to complement the non-existing global knowledge by using previous experiences encoded in the pheromones. Thus, the system learns at runtime the network topology, available resources, and the current workload of each device. The strategy seeks to cover the requirements summarized in Section 4.1 to build a decentralized, flexible, efficient, scalable task coordination that adapts to the dynamic nature of IoT devices.

### Implementation

In this section, we explain the technical specifications of the ACO strategy and what adaptations have been made to the AS so that task coordination can be more adaptive and flexible to dynamic distributed IoT clusters. In order to understand the design decisions made and changes to the traditional AS, we first explain how this type of ACO algorithm works. We then explain the technical facets of our probabilistic strategy and highlight how our system design choices attempt to address the requirements for decentralized task coordination in dynamic IoT systems.

**Traditional Ant System** The traditional AS is an iterative optimization algorithm simulating the collective behavior of ants. In each iteration, a total of  $m$  ants constructs solutions by traversing the graph and depositing pheromones on the chosen paths. After  $t_{max}$  iterations, the process ends and outputs the best-found path [6]. The algorithm's pseudo-code is provided in Chapter 2 for reference.

The AS's state transition rules guide the ants through the graph. An ant moves from one state, i.e., node, to another based on a probabilistic decision-making procedure that considers both the pheromone levels and heuristic information associated with the possible transitions. The probability of ant  $k$  moving from node  $i$  to node  $j$  is given by the Formula 4.1:

$$p_{ij}^k = \frac{(\tau_{ij}^\alpha \cdot \eta_{ij}^\beta)}{\sum_{l \in J_i^k} (\tau_{il}^\alpha \cdot \eta_{il}^\beta)} \quad (4.1)$$

This state transition rule consists of two information types. The first one is  $\eta_{ij}$ , which can be seen as the desirability of a state transition. It is the a-priori knowledge about the problem domain. On the other hand,  $\tau_{ij}$  is global knowledge, which is the amount of pheromones deposited. It is the a-posteriori knowledge based on the changes caused by ants during the iterations. The parameters  $\alpha$  and  $\beta$  control the influence of local and global knowledge.

Formula 4.2 shows how the pheromone levels on the path between nodes  $i$  and  $j$  are updated after each iteration. Moreover, a certain amount of pheromones vaporize – controlled by the parameter  $\rho$ . This pheromone evaporation mechanism prevents the algorithm from converging prematurely to suboptimal solutions. It reduces the pheromone levels on all paths, encouraging the exploration of new solution candidates.

$$\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij} + \Delta\tau_{ij} \quad (4.2)$$

$\Delta\tau_{ij}$  is described by the following Formula 4.3, which is the sum of all newly deposited pheromones by all ants in the system:

$$\Delta\tau_{ij} = \sum_{k=1}^m (\tau_{ij}^k) \quad (4.3)$$

The amount of pheromones deposited on each edge used by an ant depends on the quality of the solution  $L$ , which is typically the length of a path. The other parameter  $Q$  in the Formula 4.4 is a constant value. Shorter or more efficient solutions will result in more pheromones being deposited.

$$\Delta\tau_{ij}^k = \frac{Q}{L_k} \quad (4.4)$$

**Adapting ACO for Decentralized Task Coordination** Our ACO implementation introduces several modifications to better suit task coordination characteristics in distributed IoT systems. First, we remove the concept of multiple ants generating different solutions in one iteration. Instead, we employ only one ant, which represents the offload request. I.e., when a task is offloaded, one ant per request traverses the network looking for a suitable target node. Furthermore, we introduce the concept of two different ant types (see Figure 4.6):

1. **Forward ant.** The forward ant represents the offload request created by the origin device and carries the task and the TTL with it. It is responsible for finding a path between the origin and target nodes. When it has found a suitable node along its path that can process the task, it leaves it there and is terminated.

2. **Backward ant.** Once the target node successfully processes the task, it computes the quality pheromones and creates a backward ant. The backward ant follows the same path as the forward ant but in the opposite direction. During the traversal, it is responsible for updating pheromones based on the quality calculation at the path nodes. In contrast to Random, there is no path optimization (see Section 4.3.1) because updating the pheromone tables depends on the path length traveled between the origin and target nodes. Path optimization, on the other hand, modifies the length of the original path, which can lead to an incorrect pheromone deposition in the ACO approach.

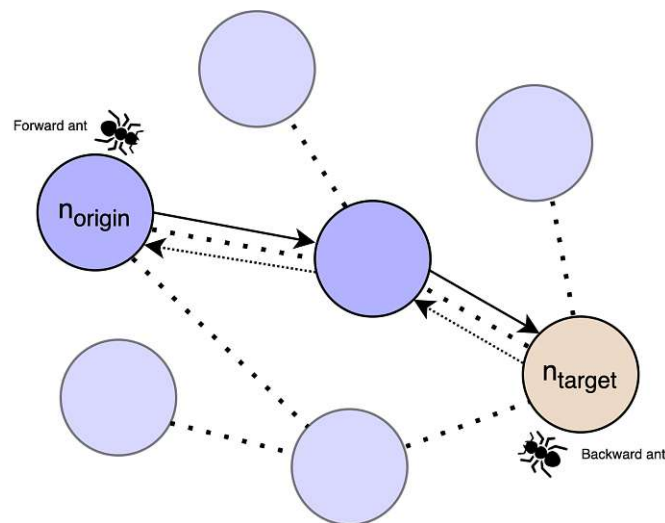


Figure 4.6: Concept of forward and backward ants.

The idea of using forward and backtrack ants originates from Michlmayr’s work, in which she employs an ACO-based algorithm to find resources in peer-to-peer networks [16, 17]. Similar to her implementation, where the forward ant carries the query for the resource, our artificial ant carries the task.

Furthermore, we eliminate iterations in our strategy. Therefore, there is one ant and only one iteration for offloading a task. These two design decisions are intended to improve scalability and reduce computation and communication overhead. Looking at a single hop between two devices, not  $t_{max} \cdot m$  messages are generated and sent, but only one, where  $t_{max}$  is the total number of iterations and  $m$  is the number of ants deployed in each iteration. On the one hand, our system scales better for more nodes. On the other hand, these modifications lead to losses in the performance of the task distribution. The strategy collects less information per task offload and thus learns slower. More on the expected characteristics of the algorithm can be found in Section 28.

Given these adaptations of the AS for our system, we can present the pseudo-code of our ACO strategy. Algorithm 4.1 shows how the ants decide which node to visit next based

**Algorithm 4.1:** The decentralized ACO strategy in pseudo-code

---

**Data:** Nodes  $nodes$ , Task  $t$

```

// Concurrent activity at each node
1 foreach  $node\ n$  do
2   initializePheromoneTables()
3   while  $n$  is running do
4     if  $n$  is overloaded then
5       createForwardAnt( $t$ )
6       while  $t_{TTL} > 0$  do
7          $n_j =$  applyStateTransitionRule()
8         goToNode( $n_j$ )
9         if  $n_j$  supports  $t_{type}$  and not overloaded then
10          computeTask( $t$ )
11          pheromones = calculateQuality()
12          createBackwardAnt(pheromones, path,  $n_j$ )
13        end
14        addNodeToPath( $n_j$ )
15      end
16    end
17    foreach backwardAnt do
18      do
19         $node =$  popNodeFromPath(path)
20        goToNode( $node$ )
21        applyPheromoneTrailUpdateRule(pheromones)
22      while  $node \neq node_{origin}$ 
23    end
24    foreach PeriodicalTimeInterval  $t_e$  do
25      applyEvaporationRule()
26    end
27  end
28 end

```

---

on the state transition rule (see Formula 4.5). The node capable of processing the task then computes the quality pheromones (see Formula 4.7), which the backward ant then uses to update the pheromones in Formula 4.9. The evaporation of the pheromones does not happen as in AS when updating the pheromones but independently every  $n$  minutes (see Formula 4.12).

Formula 4.5 shows the state transition rule, which is identical to AS's approach. The parameters  $\tau_{ij}$  and  $\eta_{ij}$  are adapted to the problem domain of decentralized task coordination. The local knowledge  $\eta_{ij}$  is exchanged during the ND protocol and indicates whether two neighboring nodes support the task type in the offload request. If so, a

higher value defined by  $\tau_{0+}$  is set. If not, the lower value  $\tau_0$  is used. The intention is to favor nodes that support the task type to reduce the offloading time and, thus, the latency and number of generated messages.

$$p_{ij}^k = \frac{(\tau_{ij}^\alpha \cdot \eta_{ij}^\beta)}{\sum_{l \in J_i^k} (\tau_{il}^\alpha \cdot \eta_{il}^\beta)} \quad (4.5)$$

$$\eta_{ij} = \begin{cases} \tau_0^+, & \text{if next node supports } T_{type} \\ \tau_0, & \text{otherwise} \end{cases} \quad (4.6)$$

Our approach calculates the quality pheromones  $Q$  at the target node, which is responsible for processing the offloaded task. The calculation is based on two primary factors: the device's queue quality and its available capacities. The device's queue quality, denoted as  $Q_L$ , measures how occupied the device's queue is. We compute this quality metric using the average queue occupation,  $L_{avg}$ , and the queue limit,  $L_{limit}$ , which is the threshold when the node is considered overloaded. The queue quality is inversely proportional to the queue occupation, which means that if the queue is less occupied, the device has more resources available, and the quality metric will be higher. The second factor in the quality pheromones calculation is the device's capacities, including communication ( $C_{comm}$ ), computation ( $C_{comp}$ ), and storage ( $C_{storage}$ ). The better these capacities, the higher the quality pheromones. The resulting formula for calculating  $Q$  is as follows:

$$Q = Q_L \cdot \frac{1}{\frac{1}{3}(C_{comm} + C_{comp} + C_{storage})} \quad (4.7)$$

$$Q_L = \begin{cases} 0, & \text{if } 1 - \frac{L_{avg}}{L_{limit}} \leq 0 \\ 1 - \frac{L_{avg}}{L_{limit}}, & \text{otherwise} \end{cases} \quad (4.8)$$

By encoding the quality pheromones in this manner, we can capture the device's current availability and inherent capacities, leading to a more informed decision-making process in the ACO algorithm. The strategy aims to select target nodes with higher availability and better resources.

Updating the pheromone tables at each node along the path is crucial for learning and guiding the search process. As mentioned, the target node calculates the quality pheromones  $Q$ , and the backtrack ant carries them along the return path to the origin node. The backtrack ant updates the pheromones at each node using the carried quality pheromones and the distance from the current node to the source node, denoted as  $Q_P$ . The rationale behind this approach is that nodes closer to the source node should receive more pheromone values, reducing the overall offload time, latency, and the number of messages used. However, the update mechanism might require several offloading processes

before converging to an optimal solution, particularly in dynamic and large-scale IoT environments.

$$\tau_{ij} \leftarrow \tau_{ij} + \Delta\tau_{ij} \quad (4.9)$$

$$\Delta\tau_{ij} = \Delta\tau_{ij}^k = \frac{Q \cdot 100}{Q_P} \quad (4.10)$$

$$Q_P = \frac{1}{\text{hopsToOrigin}} \quad (4.11)$$

Evaporation prevents the algorithm from becoming overly biased towards previously found solutions, enabling it to explore new paths and adapt to changes in the network conditions. I.e., the mechanism encourages a balance between exploration and exploitation, ensuring that the algorithm does not focus solely on well-known paths but also investigates alternative options that might offer better offloading performance. In classical AS, evaporation is achieved by subtracting a relative value during the pheromone update process. Instead of simulating the evaporation process, each node in our system performs evaporation independently, removing a relative part of the pheromone values every  $n$  minutes. The amount of pheromone reduced is controlled by the parameter  $\rho$ :

$$\tau_{ij} \leftarrow (1 - \rho) \cdot \tau_{ij} \quad (4.12)$$

**Time Complexity** In this section, we analyze the time complexity of the ACO-based task coordination strategy, considering different aspects of the algorithm, such as initialization of pheromones, quality pheromones calculation, applying state transition rules, updating of pheromones, and the evaporation process.

- **Pheromones Initialization** When two devices are connected, the initialization involves updating the pheromone tables for the connected edges. This process has a time complexity of  $O(n)$ , where  $n$  is the number of connected edges.
- **Quality pheromones calculation.** The calculation of quality pheromones at the target node is a constant-time operation, resulting in a time complexity of  $O(1)$ .
- **State transition rule.** The selection of the next node depends on the state transition rule, which considers the connected edges and the supported task types of the selected node. The time complexity of this process is  $O(n \cdot m)$ , where  $n$  is the number of connected edges, and  $m$  is the number of supported task types of the selected node.

- **Pheromones update.** Updating pheromones with the backtrack ant involves traversing the connected edges and updating the pheromone values accordingly. This results in a time complexity of  $O(n)$ , where  $n$  is the number of connected edges.
- **Evaporation.** The evaporation process involves updating the pheromone values for each connected edge, and each task type the node has heard of. This results in a time complexity of  $O(n \cdot m)$ , where  $n$  is the number of connected edges, and  $m$  is the number of task types.

**Parameters** Table 4.3 lists all the parameters the framework needs to run the ACO strategy.

Name	Description
$\tau_0$	Initial default amount of pheromone deposited. Can be any decimal value between 1 and 100.
$\tau_0^+$	Initial default amount of pheromone deposited, if two adjacent nodes support min. 1 shared task type. Can be any decimal value between 1 and 100.
$\alpha$	Controls influence of $\tau$ (a-posteriori knowledge). Can be any positive integer value.
$\beta$	Controls influence of $\eta$ (a-priori knowledge). Can be any positive integer value.
$\rho$	Controls amount of pheromones being evaporated. Can be any decimal value between 0 and 1.
evaporationFrequency	Controls how often pheromones being evaporated. Can be any CRON expression.
offloadedTasksCleanUp	Controls how often offloaded tasks being removed in hours. Can be any positive integer value.

Table 4.3: ACO strategy's parameters.

### Characteristics

By utilizing forward and backward ants instead of multiple ants in multiple iterations, our approach significantly reduces the number of messages exchanged among nodes, thereby relieving latency and bandwidth constraints. Additionally, it shortens the overall offload duration, resulting in smaller queue sizes. However, that comes with the cost of the accuracy of task coordination. Due to the missing ants and iterations, the system needs more time to learn, which harms the performance of the task distribution. We expect

worse performance results than a traditional AS algorithm, but we predict much better scalability.

Furthermore, our approach focuses on a small computational overhead for IoT devices by using straightforward rules for pheromones calculation, updates, and state transition rules. It enables the nodes to focus on processing tasks rather than consuming resources on the coordination algorithm.

Finally, encoding the ACO-based task coordination approach inherently encodes the dynamic nature of IoT devices. Pheromone updates and evaporation processes facilitate the algorithm's adaptation to device availability, capacity, and network conditions changes. However, we expect degraded performance for node failures that rejoin the network because the accumulated experience, i.e., pheromones, are lost. On the one hand, this penalizes nodes that crash often and are thus not reliable. On the other hand, the connections of nodes that are rarely dropped also have to be retrained.

### 4.3.3 Gossips

#### Motivation

The third and last decentralized task coordination strategy is titled Gossips, based on the functionality of the Gossips protocol. Compared to the previous two algorithms, this methodology works in two stages. In the first stage, a naive Gossips protocol collects information from the cluster. The origin distributes offload requests to its neighbors, which in turn forward them to their neighbors. Those nodes that can accept the offload request send a reply to the origin. In the second phase, the origin node receives all this data and chooses the most promising target node to process the task.

The Gossips protocols are a popular approach for sending information in distributed systems where global knowledge is not easily attainable. It is particularly well-suited for environments that require adaptability to changes, as the strategy floods the system with messages, ensuring that new knowledge about the cluster is obtained with each request. One of the motivations behind using the Gossips-based task coordination strategy is to compare its performance with that of the ACO approach. This comparison helps assess the trade-off between scalability and task coordination accuracy. We expect Gossips to perform coordinating tasks more accurately due to the comprehensive information the strategy gathered during the first phase. However, there will likely be scalability issues as the number of messages sent overwhelms the network. On the other hand, ACO is more scalable but may not be as accurate as Gossips, as it lacks knowledge about the cluster. Investigating these trade-offs provides valuable insights into the performance of these two coordination strategies in the context of distributed IoT systems.

#### Implementation

As mentioned above, the Gossips approach can be divided into two main phases: the Gossips Discovery Phase and the Gossips Offload Phase. The strategy's pseudo code

provided in Algorithm 4.2 serves as a reference for understanding the implementation.

---

**Algorithm 4.2:** The decentralized Gossips strategy in pseudo-code

---

**Data:** Nodes  $nodes$ , Task  $t$

```

// Concurrent activity at each node
1 foreach node  $n$  do
2    $messages = \emptyset$ 
3    $responses = \emptyset$ 
4   while  $n$  is running do
5     if  $n$  is overloaded then
6       while  $t_{TTL} > 0$  do
7         foreach neighbor of  $n$  do
8           | spreadDiscoveryGossips()
9         end
10      end
11     end
12     foreach received discovery gossip  $d$  do
13       if  $d \in messages$  then
14         | stopSpreadingDiscoveryGossips( $d$ )
15       end
16       if  $n$  supports  $t_{type}$  and not overloaded then
17         | sendGossipResponse( $d$ )
18       end
19       addToMessages( $d, messages$ )
20     end
21     foreach received gossip discovery response  $r$  at  $n_{origin}$  do
22       | addToResponses( $r, responses$ )
23        $request = applyQualityResponseSortingRule(responses)$ 
24       | offloadTask( $request, t$ )
25     end
26   end
27 end

```

---

**Gossips Discovery Phase** The primary goal of the Gossips Discovery Phase is to contact as many nodes as possible to find available target nodes for task offloading. The stage can also be regarded as a way of assessing the current status of the cluster in order to get a picture of the available devices and their resources. To this end, the origin node starts spreading messages, so-called Gossips discovery messages, to all its neighbors, which in turn propagate these messages to their neighbors. When a node receives a gossip discovery message and can process the task, it sends a discovery response along the path to the origin node. The first phase continues until the same gossip message is received by

a node or the discovery TTL passed, determined by the origin node at the beginning. At this point, the gossip propagation for that discovery request stops. The design decision behind this naive Gossips protocol implementation is gathering network knowledge rather than relying on probabilities, as in the Random and ACO strategies. This deterministic approach aims to build extensive knowledge about the cluster and its available resources.

**Gossips Offload Phase** In the Gossips Offload Phase, the following three steps are performed: The origin node collects all received discovery response messages, which contain information about the current queue capacity and the on-board hardware resources of the target node. In the second step, the strategy uses this information and the path length to calculate which target node appears to be the most attractive for offloading the task. Formula 4.13 is used for this purpose. In the last and third step, the offload request is sent to the selected target.

$$Q = \left(1 - \frac{L_{avg}}{L_{limit}}\right) \cdot \frac{1}{\frac{1}{3}(C_{comm} + C_{comp} + C_{storage})} \cdot Q_P \quad (4.13)$$

where  $L_{avg}$  represents the average queue occupation,  $L_{limit}$  denotes the queue limit, and  $Q_P$  path quality (see Formula 4.11).

**Path Optimization** As in Random, the strategy tries to shorten the discovery response path by looking if other devices along the path are connected to the origin node. The motivation and operation are identical to those in the Random algorithm. For this reason, we refer to Section 4.3.1, provided in the Random task coordination strategy section.

**Time Complexity** This section provides a time complexity assessment of the Gossips task coordination strategy. The complexity of the Gossips strategy can be broken down into two main components: spreading and forwarding gossip discovery messages and selecting the target node to offload the task.

- **Spreading Gossips Discovery Messages.** The complexity of spreading and forwarding Gossips discovery messages can be represented as  $O(n)$ , where  $n$  is the number of connected edges at a node.
- **Selecting the Target Node.** The selection process involves sorting the received response messages and choosing the highest-ranked node as the target for offloading the task. This sorting process has a complexity of  $O(n \cdot \log(n))$ , where  $n$  is the number of received gossip discovery response messages.

**Parameters** Table 4.4 lists all the parameters the framework needs to run the Gossips strategy.

Name	Description
<code>spreadingTTLFactor</code>	Constant controlling duration of discovery phase. Can be any positive decimal value.
<code>offloadedTasksCleanUp</code>	Controls how often offloaded tasks being removed in hours. Can be any positive integer value.

Table 4.4: Gossips strategy's parameters.

### Characteristics

One of the key characteristics of the Gossips strategy is its fast adaptation to changes in the network. As the strategy floods the network with messages, it constantly acquires new knowledge about the state of the network, including the availability of resources and nodes' workload. The continuous update of information allows the strategy to adapt quickly to changes in the network, such as the addition or removal of nodes.

The deterministic nature of the Gossips strategy is another essential characteristic. By attempting to contact as many nodes as possible and building a comprehensive network knowledge, Gossips has a higher probability of successfully offloading tasks. This deterministic approach contrasts with probabilistic strategies, which rely on random choices and may not always find the best target nodes for offloading tasks. While the deterministic nature of the Gossips strategy contributes to its accuracy in task coordination, it also leads to scalability issues, as the network can become flooded with messages, causing increased latency and bandwidth consumption. Furthermore, the offload duration may increase because the strategy consists of 2 phases, and the origin has to wait until the TTL has passed before deciding on a target node.

The trade-off between scalability and task coordination accuracy becomes evident when comparing the Gossips strategy with the ACO-based approach. While the former strategy may provide more accurate task coordination due to its deterministic nature and constant updates of network knowledge, the ACO strategy can offer better scalability, as it relies on fewer messages exchanged within the network. However, ACO may not be as accurate as Gossips in task coordination, as it lacks complete knowledge about the cluster.

## 4.4 Communication

Communication plays an essential role in distributed IoT systems, along with coordination. A solid communication approach enables devices of a decentralized IoT cluster to interact and collaborate. However, some challenges must be addressed when designing reliable communication for the distributed IoT model. Many use cases of IoT applications, especially time-sensitive ones, require low-latency networks to perform real-time decision-making. Furthermore, communication methods must pay attention to bandwidth. The bandwidth of IoT networks is often limited, which has a negative impact on data-

intensive applications. Communication for distributed IoT systems must keep the network manageable by minimizing the number and size of messages. Another critical issue is the heterogeneity of IoT devices. Due to the large number of devices with different hardware and architecture on-board, not all communication technologies are supported. Last but not least, the dynamic nature of IoT devices should be remembered: The network topology can constantly change due to mobility. IoT devices can leave or (re-)join a network at any point in time.

All these aspects must be considered when designing efficient communication in distributed IoT systems. Zeqo's work looks at this topic in detail and describes what communication looks like for our framework [23]. On the other hand, we give a high-level overview of framework communication in the following paragraphs. While our work is responsible for the design and implementation of decentralized task coordination, communication deals with the exchange of information in the cluster, the construction of minimal messages, the bootstrapping of clusters, and how the cluster topology is dynamically adjusted based on the strategy metrics.

### Protocols

Different protocols have been developed for intra-cluster communication. They are responsible for encoding and decoding messages and, depending on the message type, invoke the correct logic in the framework. There are protocols for connecting devices, applying different task coordination strategies, and running SA (see Section 4.4). The goal of these protocols, which use TCP, is to reduce the message size but, at the same time, provide enough information for the devices to find their way around despite the limited local knowledge of the graph. Using the protocols, devices exchange messages built in the JSON format for easy parsing. A protocol handler in the framework decides at runtime which protocol to invoke for which message based on the message type.

### Node Discovery

ND aims to form clusters by enabling IoT devices to discover and connect to each other on the network. In the framework, each device defines for itself with which IP address it is discoverable. With this IP address, the node accepts new connections. In addition, each device defines an IP range in which it searches for new connections to devices in the cluster. It iterates through the range and tries to establish a connection. Additionally, each IoT device specifies the minimum number of open connections. E.g., if the value is set to  $n$ , the device tries to connect to new devices until it reaches the threshold. Once connected to  $n$  devices, it stops looking for new nodes in the cluster. However, it can still accept new incoming connections. When two devices are connected, they exchange local node profiles as part of the ND protocol. The local node profile consists of the device resources (processing, storage, and communication capabilities) and the task types that the device supports (see Section 4.2.1). The information is used to build local knowledge about the network graph. After the ND protocol has been successfully completed, the devices can exchange further messages.

### Self-Actualization

SA dynamically adjusts the cluster topology to enhance task coordination performance. Nodes do not connect in a cluster according to specific criteria but simply whether they can establish a connection within the IP range. I.e., the topology is randomly constructed. The performance of the task coordination suffers from this structure because there exist nodes that cannot assist their neighbors. However, they could support participants of the swarm, which are further away. To overcome this limitation in the system, SA tries to detect these nodes and disconnect them from the other devices. The method hopes that the disconnected node will connect to other nodes where its resources are more needed. SA aims to place devices that can support each other closer within the swarm.

The task coordination strategies store a specific value for each outgoing connection to other nodes, which indicates how attractive this connection is (see Chapter 4.3). SA takes this local knowledge about the cluster and calculates the weakest links. This computation occurs at each node and indicates which links should be disconnected. However, the disconnection of the devices must not happen all at once because it is still possible that a task was offloaded using these nodes. If the connection is terminated, there is a risk that no response can be sent back. For this reason, the SA process takes place in two phases. In the first stage, the so-called avoid process, the nodes are informed that they should no longer contact specific nodes for offloading tasks because these nodes will soon disconnect. After this stage, the neighbors remember the detached nodes by storing them in a block list. In this 2<sup>nd</sup> phase, connection requests from blocked devices are denied to prevent the same swarm topology from being constructed. The block phase forces disconnected nodes to search for new neighbors. After some time, the marked devices are removed from the list.

SA runs with the task coordination strategies ACO and Gossips but not with Random because the topology changes happen randomly with it as well.

## 4.5 System Monitoring

For the master thesis, a central monitoring system was implemented to store information about the devices in the swarm, their communications, and the performance metrics of the strategies. Devices send this information to the monitoring application, which validates and stores it. The retrievable data is necessary for the evaluation of the framework.

The system consists of three main components, which are illustrated in Figure 4.2:

1. **Message Broker.** To ensure that devices are not blocked when sending data to the monitoring component, RabbitMQ<sup>4</sup> is used as a message broker. Due to the asynchronous communication, the IoT devices are not temporally coupled by the monitoring application.

---

<sup>4</sup><https://www.rabbitmq.com/>

2. **Monitoring Backend.** The Spring Boot application<sup>5</sup>, written in Java<sup>6</sup>, acts as a consumer for the message broker and a client for the database. It validates, processes, and stores received messages.
3. **Database.** The monitoring system does not use a traditional database for storing data, but a graph database, in this case, Neo4j<sup>7</sup>. This choice of technology is because the IoT system can be represented as a graph. The devices are the nodes, while the connections can be abstracted as edges. Furthermore, requests and other communication between two machines can be represented as edges. The advantages are manifold: There is an out-of-the-box presentation of the topology in Neo4j. Nodes and edges can be represented differently. There is the option to view the swarm over time. Moreover, the queries to form metrics from the data are more intuitive and straightforward.

In order to be able to evaluate the framework, specific metrics such as the number of generated messages in the system, the number of successfully processed offloaded tasks, etc. (see Chapter 5). The devices send many events, which are then stored by the monitoring system to calculate these metrics. These include new and closed connections between nodes, offload requests, information on whether a device can accept an offload request and thus process the task, or whether the TTL has passed. Furthermore, other events are stored, such as whether the offloaded task could be processed and thus returned and various SA and gossip messages. Furthermore, every 2 minutes, each device sends information about the queue occupation's min, max, average, standard derivative, and variance value, which the device collects internally every 10 seconds.

---

<sup>5</sup><https://spring.io/>

<sup>6</sup><https://www.java.com/en/>

<sup>7</sup><https://neo4j.com/>



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar  
The approved original version of this thesis is available in print at TU Wien Bibliothek.

# Evaluation

This chapter aims to evaluate the performance of the three task coordination algorithms presented in this thesis, Random, ACO, and Gossips, in the context of decentralized IoT systems. In doing so, we will focus on RQ2 and consider how the ACO algorithm impacts task coordination's performance compared to the remaining approaches. To this end, we run tests on servers on which VMs represented as IoT devices run the framework respectively. By running tests with different numbers of devices in each static and dynamic environment, we get a solid overview of how the strategies perform. We compare the task coordination implementations through performance metrics such as load distribution of the queue, service time, and hops per hit.

The remainder of this Chapter is organized as follows: In Section 5.1, we discuss the testing setup, including the metrics used for the evaluation, the static and dynamic environment tests, and the device and task emitter configurations. Section 5.2 provides an overview of the testbed, detailing the server and virtual machine specifications. Finally, Section 5.3 presents the test results evaluating and comparing the performance of the strategies.

## 5.1 Experiments Setup

### 5.1.1 Performance Metrics

To effectively compare the performance of the three task coordination strategies, we rely on a set of metrics emanated from related work [5, 7]. These metrics allow us to analyze various aspects of the strategies and are defined as follows:

1. **Load Distribution (LD).** The average queue occupation measures load distribution. The metric reflects the system's efficiency and aims balanced load, where the queue is neither empty nor too full.

2. **Service Time (ST).** It measures when the origin node sends an offload request for a task until it receives a response back. I.e., how long does the origin have to wait until it can regard the offloaded task as finished? It is expressed in seconds.
3. **Hops per Hit (HPH).** The metric measures the hops required to find the target node. It indicates how often another node must be contacted until the target node is found.
4. **Hit Miss Ratio (HMR).** The ratio shows how often a target node was found divided by the number of times a target node was not found, but the offload request was lost (e.g., if TTL had already passed).
5. **Guarantee Ratio (GR).** The ratio is calculated by dividing the number of offloaded tasks by the number of returned tasks (from the origin node to the target node and back to the origin node).
6. **Amount of Messages (AM).** The metric counts the number of messages the active strategy produces while coordinating tasks during runtime.
7. **Messages per Request (MPR).** The metric calculates the number of messages produced by the active strategy for a single offload request for a task.

In this thesis, we focus on the first three metrics. In contrast, the rest of the metrics are in the area of communication and are therefore examined in more detail in Zeqo's work [23].

### 5.1.2 Configurations

#### Environment Simulation

To thoroughly evaluate the performance of the task coordination strategies, we conducted experiments in two types of environments: static and dynamic. These environments and varying numbers of devices allow us to simulate the complex nature of decentralized IoT systems and assess the scalability and adaptability of the strategies.

In the static environment, the network topology remains unchanged throughout the experiment. This type of environment enables us to assess the performance of the strategies in a stable and controlled setting. On the other hand, the dynamic environment simulates decentralized IoT systems' mobility and ever-changing nature. In this environment, devices can fail and rejoin the network. Every 15 minutes, we calculate a probability to determine if a node will fail based on each node's characteristics. If the probability is above a certain threshold, the device fails. However, devices also can recover and rejoin the cluster. Every minute, a probability is calculated for a failed device to recover, and if it exceeds the recovery threshold, the device rejoins the network. This dynamic environment allows us to evaluate how well the strategies can adapt to network changes and maintain performance.

In addition to the two types of environments, we also varied the number of devices in each run. The experiments included 10, 25, 50, or 100 devices, each with different characteristics to simulate a mix of more robust nodes and resource-constrained devices. This heterogeneity of devices helps us assess how well the strategies can handle diverse resource capacities in the network. Section 5.1.2 explains how we simulate the device types and properties.

By combining static and dynamic environments with varying numbers of devices, we can better evaluate the strategies regarding scalability and adaptability. For example, we can investigate whether a strategy can scale with an increasing number of devices or maintain its performance in a dynamic environment with nodes frequently failing and rejoining.

To summarize, our test design is as follows: For each strategy, we run tests once in a static environment and once in a dynamic environment. Each of these tests then runs with 10, 25, 50, and 100 devices, respectively. This results in a total of 24 experiment runs. Each run lasts a total of 60 minutes.

### Device Simulation

Device type	Task types	Queue capacity	$C_{comm}$	$C_{comp}$	$C_{storage}$	Fail probability	Recover probability
$s_1$	$T_1, T_2$	10	1	1	1	0.1	0.4
$s_2$	$T_3, T_4, T_5$	15	1	2	1	0.1	0.4
$m_1$	$T_1, T_4$	10	1	2	2	0.2	0.3
$m_2$	$T_5$	5	2	2	2	0.2	0.3
$m_3$	$T_2, T_3$	10	2	2	3	0.2	0.3
$w_1$	$T_1$	5	3	3	3	0.3	0.2
$w_2$	$T_2$	5	3	2	3	0.3	0.2
$w_3$	$T_3$	5	3	3	3	0.3	0.2
$w_4$	$T_4$	5	3	3	3	0.3	0.2
$w_5$	$T_5$	5	2	3	3	0.3	0.2

Table 5.1: Device types and their characteristics.

Because we run up to 100 devices simultaneously in a test run, we decided to virtualize them by deploying VMs. Through different device parameters, which are built into the framework, we achieve heterogeneity in the system. As described in Section 4.2.1, devices can accept different values for computation, communication, and storage capabilities. Hence, a grouping of devices with different capabilities is achieved: Strong devices, which are abbreviated with the letter  $s$ ; mid-strong devices ( $m$ ); and resource-constrained devices, which are the weakest with respect to resources in the cluster. They are also referred to as weak devices ( $w$ ) in this work. We use the notation  $s_i$  to show that this device is a strong node named  $i$ . The same is true for  $m$  and  $w$  machines.

Strong devices can not only process more tasks in a given time but also have more emitters. Having more emitters has two consequences in our system: 1.) the device supports more task types because each emitter has a different task type, and the number of different emitters defines how many task types a device can produce and support at

the same time. 2.) The queue capacity grows with the number of task types. Per type, the device gains five more places in the queue until it is considered to be overloaded. Table 5.1 summarizes the relationships. One emitter means one task type, which in turn means five places in the queue. We use the notation  $T_i$  to indicate the task type sent by the emitter. E.g., a device with two task types  $T_i, T_j$  has thus simultaneously two emitters for the respective tasks.

Furthermore, the device types have different minimum numbers of open connections to other cluster participants. The idea is that more potent devices should be connected to more nodes because it has more resources to help. The number of deployed nodes in a test run also affects the minimum number of neighbors, summarized in Table 5.2

#Nodes	#min conn.		
	<i>s</i>	<i>m</i>	<i>w</i>
10	4	3	2
25	5	4	3
50	6	5	4
100	7	6	5

Table 5.2: Number of nodes and minimum connections per device type.

Finally, Table 5.3 summarizes the number of each device type in the test runs. The target is an approximate relative ratio of 20% of type *s*, 30% of type *m*, and the remaining 50% have type *w*. Despite different node numbers, these proportions give us similar heterogeneity in each test run, improving experiment comparability.

#Nodes	# <i>s</i> <sub>1</sub>	# <i>s</i> <sub>2</sub>	# <i>m</i> <sub>1</sub>	# <i>m</i> <sub>2</sub>	# <i>m</i> <sub>3</sub>	# <i>w</i> <sub>1</sub>	# <i>w</i> <sub>2</sub>	# <i>w</i> <sub>3</sub>	# <i>w</i> <sub>4</sub>	# <i>w</i> <sub>5</sub>
10	1	1	1	1	1	1	1	1	1	1
25	2	3	2	2	3	2	2	3	3	3
50	5	5	5	5	5	5	5	5	5	5
100	10	10	10	10	10	10	10	10	10	10

Table 5.3: Device distribution in different cluster sizes.

### Task Emitter Simulation

In our test design, the emitters, which dispatch task types  $T_1$  to  $T_5$ , are simulated with the help of test data. As described in Section 4.2.4, a task emitter outputs a task with the type and its processing time at a specific frequency.

For the processing time, we use test data taken from Müller’s work [18]. We read the data in milliseconds and multiply them by a factor of  $10^{-2}$  for our system. I.e., we treat the values as seconds. The average value of the resources is multiplied by processing time to simulate the different device capacities. This value will be higher for a node with

poorer resources than for more powerful machines. We use the data set *object detection* for all task types:  $T_1$  is populated with data from *xeon*, where the function is the shortest. The other task types,  $T_2$  and  $T_3$ , are based on *nx*.  $T_4$  and  $T_5$  use *nano* to obtain the processing time of the tasks.

For the arrival rate of the tasks, we have used the request generator from the tool Edge Runner<sup>1</sup>. As seen in Figure 5.1, the task types have different times when emitted to increase the heterogeneity.  $T_1$  and  $T_3$  appear most often in the system, while  $T_4$  and  $T_5$  are the task types emitted least often. Nonetheless, their processing time is the longest. At any time, an emitter can send out precisely one task, not several. Combined, however, the emitters can eject several tasks at once. Due to the overlap of arrival rates, IoT devices may produce many tasks at once in a short period. The process can overload the individual cluster members or the entire system. It will be interesting to see how task coordination deals with this in the evaluation.

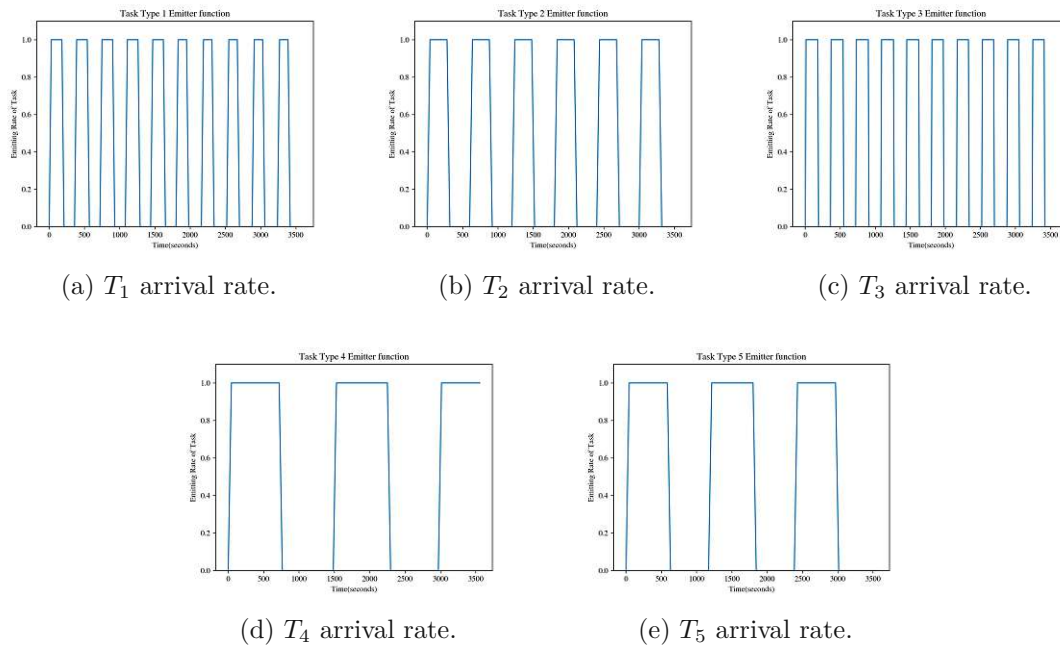


Figure 5.1: Arrival rates for different task types.

### Parameters of the Framework

The following Table 5.4 provides the values of the system parameters used for the experiments.

<sup>1</sup><https://github.com/edgerun/request-generator>

Name	Value
offloadedTasksCleanUpHours	1
initialDelayMs	600,000
fixedRateMs	600,000
blockListMinutes	15
pendingTasksCleanUpMinutes	4
$\tau_0$	50
$\tau_0^+$	100
$\alpha$	2
$\beta$	1
$\rho$	0.3
thresholdSA	0.2
evaporationFrequency	0 */2 * * * *
spreadingTTLFactor	0.5
thresholdSA	1.0
statisticsCron	*/10 * * * * *
queueCron	0 */2 * * * *
functionTimeMultiplyFactor	0.01
timeToLiveFactor	7

Table 5.4: Framework’s parameter values used for the experiments.

## 5.2 Testbed

In this section, we describe the testbed used for conducting experiments to evaluate the performance of the task coordination strategies. The testbed consists of three server machines and multiple VMs running on these servers. The hardware and software specifications of the servers and VMs are provided in detail below.

### 5.2.1 Server Specifications

The experiments were conducted on three server machines, each with the following specifications summarized in Table 5.5:

Component	Specification
Server Model	Dell PowerEdge R640 Server
Processor	2x Intel Xeon Gold 6152 2.1 GHz, 22 Cores/44 Threads
Memory	12x 16 GB-RDIMM, 2.666 MT/s, Dual Rank
SSD	2x 960 GB SATA. Mixed Use 6 Gbp/s 512 2.5" Hot-Plug AG-Drive, 3 DWPD, 5256 TBW
Hard Drive	4x 1.8TB 10K RPM SAS 12Gbit/s 512e 2.5" Hot-Plug
Controller Card	1x BOSS Controller Card with 2x M.2 Sticks 480 GB (RAID 1), LP
RAID Controller	1x PERC H740P RAID Controller, 8GB NV Cache, Mini card
Power Supply	1x Redundant Power Supply (1+1), dual, Hot-Plug-capable, 750 W
Network Interface Card	1x Broadcom 57416 two Ports 10Gbit/s Base-T + 5720 two Ports 1Gbit/s Base-T, rNDC

Table 5.5: Server hardware specification.

### 5.2.2 VM Specifications

In total, 100 VMs were deployed on the server machines with the following hardware specifications:

Component	Specification
RAM	3072 MB
vCPUs	2
Disk Space	20 GB
Disk Format	RAW

Table 5.6: VM hardware specifications.

The virtual machines are required to run the following software for the framework:

- Operating System: Rocky Linux 9.1 (minimal installation)<sup>2</sup>
- Java Runtime Environment: OpenJDK 11<sup>3</sup>

The testbed was designed to provide a reliable and controlled environment for evaluating the performance of the task coordination strategies under various conditions, as described in the previous sections.

## 5.3 Results

In this chapter, we aim to analyze the performance of the three task coordination strategies: Random, ACO, and Gossips, regarding LD, ST, and HPH. By studying these metrics under various settings, such as clusters with different node sizes and environments (static and dynamic), we strive to answer how the decentralized ACO strategy impacts the task coordination performance compared to the Random and Gossips approaches.

<sup>2</sup><https://rockylinux.org/>

<sup>3</sup><https://openjdk.org/>

It allows us to draw meaningful conclusions on the strengths and weaknesses of each strategy, ultimately helping us determine if ACO is a suitable approach for our use case and guiding future research in decentralized task coordination for IoT systems.

This evaluation is vital for understanding the trade-offs between these strategies and identifying potential future work and research directions in the field of decentralized task coordination. While our emphasis is on the metrics mentioned above, it is worth mentioning that the other metrics related to communication performance are studied in Zeqo's work [23]. These metrics combined provide a more comprehensive view of the system's behavior. Nevertheless, for the scope of this thesis, we concentrate on evaluating task coordination. Additionally, we would like to remark that we do not include all the diagrams for every possible configuration in this section. However, interested readers can find the complete set of plots in the attached link in the appendix.

We summarize the observations of the three metrics in terms of possible application cases in Chapter 6.

### 5.3.1 Load Distribution

This section evaluates the LD metric, defined as the system's efficiency in resource utilization and calculated as the average queue occupation. The primary aim of this metric is to assess the balancing of load across the system. A balanced LD ensures that tasks are offloaded to appropriate target nodes, preventing overload on specific nodes while others remain underutilized. It translates into a more efficient system that can handle tasks effectively and maintain high performance under varying workloads.

We use the term queue capacity to describe how many tasks must be in the queue before subsequent tasks must be offloaded. Queue occupation means the total number of tasks in the queue, both local tasks and tasks that will be offloaded. If the queue occupation exceeds the capacity, we call the device overloaded. The average queue occupation in our system is the LD described above.

In the first scenario with ten nodes, Figure 5.2 shows an underuse of the queue in all three strategies and both environments when the queue capacity has the two smaller values, five and ten. From this, there is room for optimization of the strategies concerning smaller queue capacities. Nevertheless, all three algorithms have a slight error rate and predictable patterns. Looking at the largest queue capacity of 15 in our static system, we can see that all three approaches have a negative relationship and better queue utilization. ACO has the slightest fall, which could indicate that the offload requests are routed to the resources in a more self-learning manner than in the remaining two strategies. Random reveals the most errors in the dynamic setting and a slightly higher positive relationship than ACO. Gossips, on the other hand, reduces the predicated queue occupation over time. However, the comparisons between the strategies in the dynamic setting are difficult to make because the different dynamics strongly impact the performance.

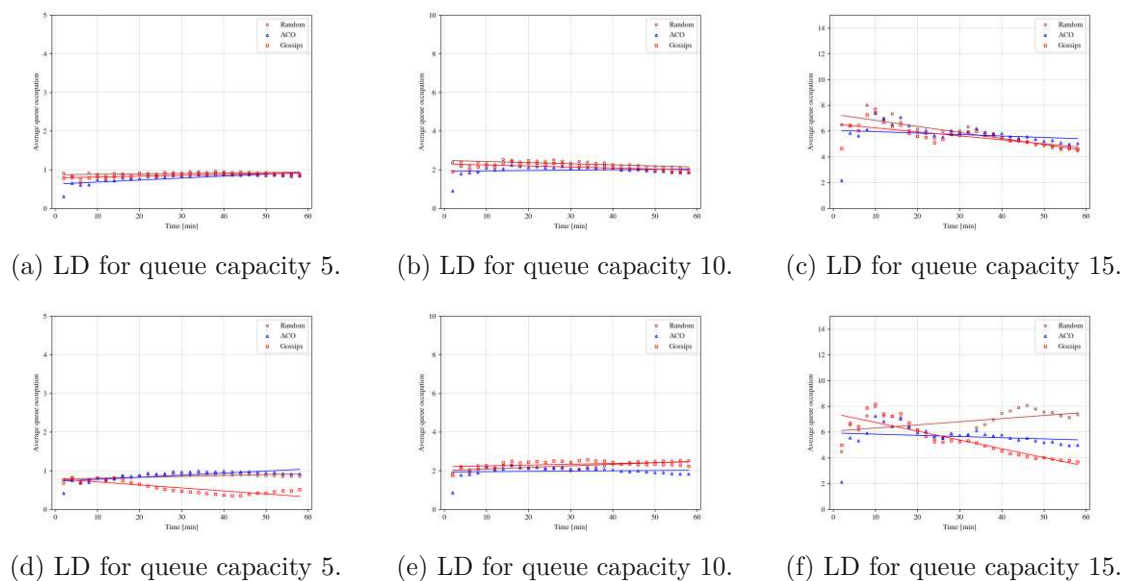
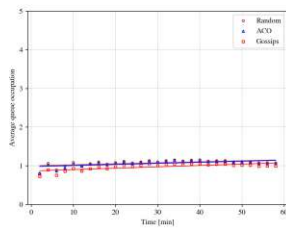


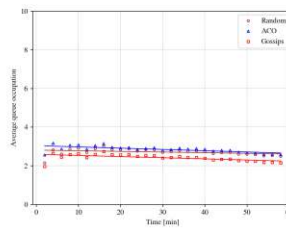
Figure 5.2: LD in static (1<sup>st</sup> row) and dynamic (2<sup>nd</sup> row) environments with ten devices of different queue capacities.

Next, we consider the LD results with cluster sizes of 25 nodes, summarized in Figure 5.3. As observed before, all three strategies again exhibit predictable patterns with few errors for queues with a capacity of five or ten tasks. With the dynamic environment, there is a more positive relationship than when the experiment is statically configured. Generally, the LD values are slightly larger than those with ten nodes, but still, the devices are underutilized. With a queue capacity of 15, we can observe in the static environment that Random, ACO, and Gossips show slight negative relationships with, as in the last paragraph, more errors at the beginning of the test runs. When the algorithms act in a dynamic evolution, the prediction seems that Random and ACO use the queue better, but it looks like they overload the devices over time. Conversely, Gossips shows a stable linear regression line but with the prospect that the queue is underutilized. Looking at all the graphs, ACO makes the best use of the queue with 25 devices. Only in the dynamic setup with devices having a queue capacity of 15, the nodes do not manage to keep up.

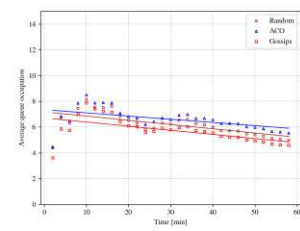
## 5. EVALUATION



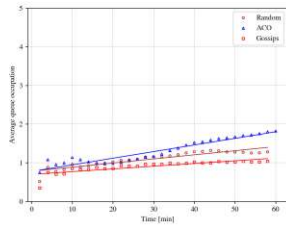
(a) LD for queue capacity 5.



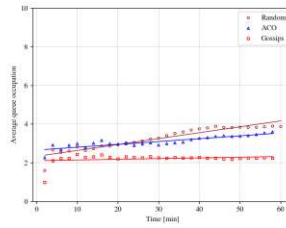
(b) LD for queue capacity 10.



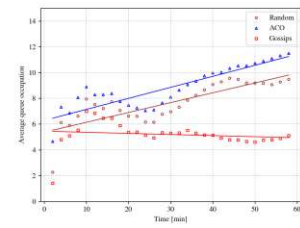
(c) LD for queue capacity 15.



(d) LD for queue capacity 5.

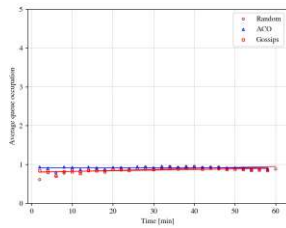


(e) LD for queue capacity 10.

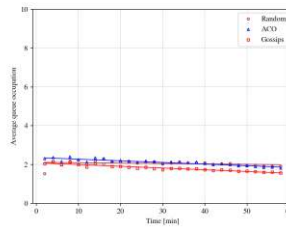


(f) LD for queue capacity 15.

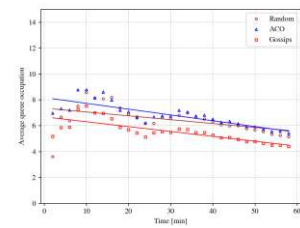
Figure 5.3: LD in static (1<sup>st</sup> row) and dynamic (2<sup>nd</sup> row) environments with 25 devices of different queue capacities.



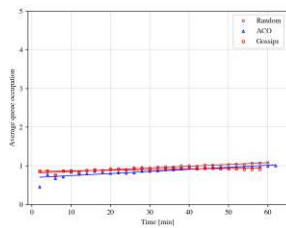
(a) LD for queue capacity 5.



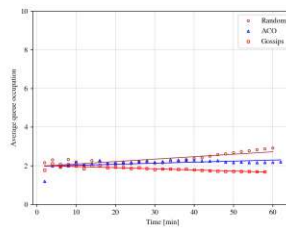
(b) LD for queue capacity 10.



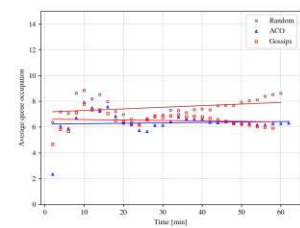
(c) LD for queue capacity 15.



(d) LD for queue capacity 5.



(e) LD for queue capacity 10.



(f) LD for queue capacity 15.

Figure 5.4: LD in static (1<sup>st</sup> row) and dynamic (2<sup>nd</sup> row) environments with 50 devices of different queue capacities.

Because the results for 50 nodes (Figure 5.4) show the same pattern as those for 25, we leave the interpretation to the reader. We now consider the LD for a cluster size of 100

nodes, which better shows how each strategy performs for this metric. In Figure 5.5, we can observe the first feature: The static Random approach exhibits stronger deviations from the regression line and grows slightly at a queue capacity of ten. With a queue capacity of 15, this trend is amplified: Random’s task coordination loses performance, and the prediction states that the devices are overloaded after one hour. We conclude that Random can only cope with smaller queue capacities for large cluster sizes. Gossips shows the smallest LD values, while ACO has the higher value but never overloads the devices, which can be concluded as a solid task coordination mechanism.

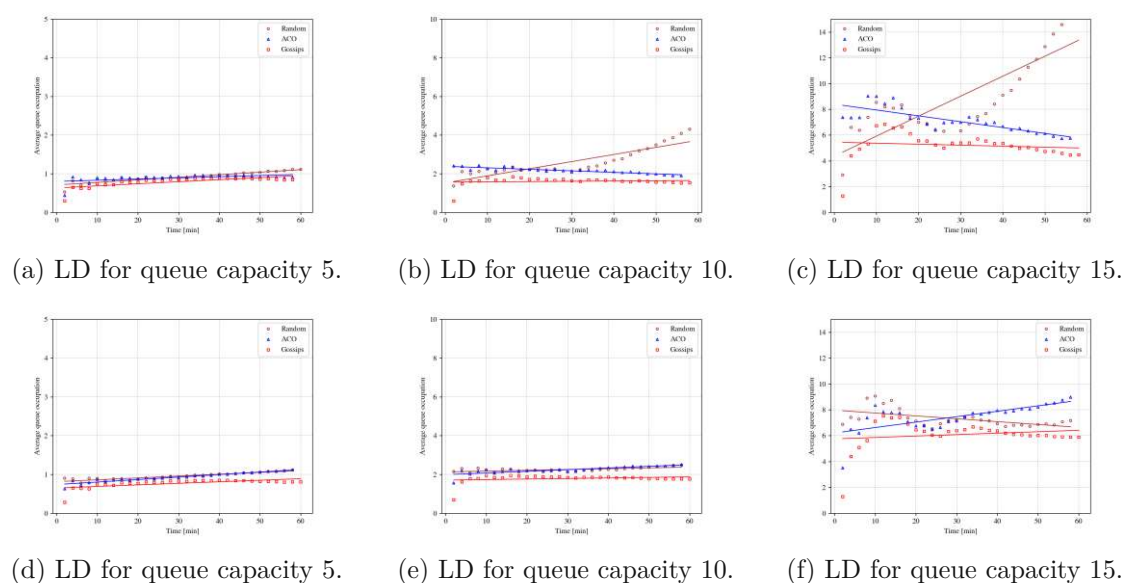


Figure 5.5: LD in static (1<sup>st</sup> row) and dynamic (2<sup>nd</sup> row) environments with 100 devices of different queue capacities.

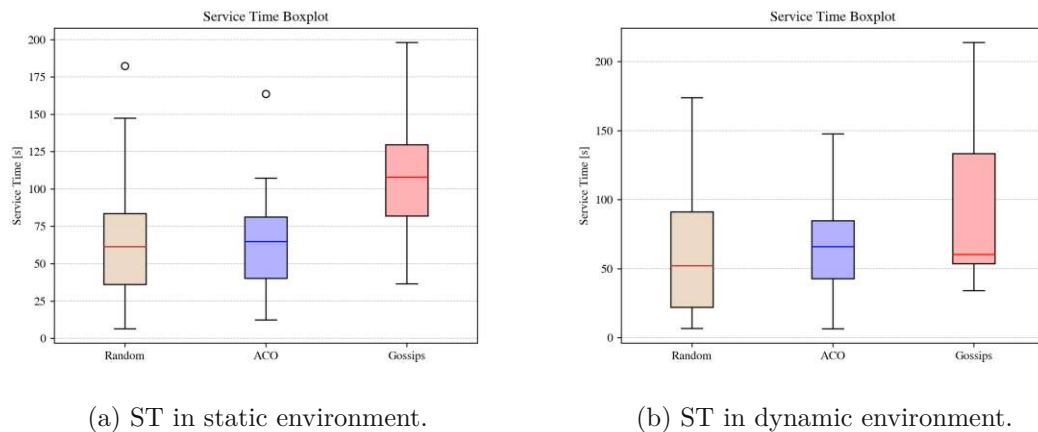
In summary, ACO consistently demonstrates the most effective utilization of queue resources across various scenarios and cluster sizes, likely due to its self-learning nature. Random shows that with large cluster sizes and larger queue capacities, it overloads the devices and thus degrades the overall system performance. The Gossips strategy shows the most stable values without the risk of overloading devices, but the queue utilization is in the lower range. In general, all three approaches have underutilization at the five and ten queue capacities, which can be improved in future works.

### 5.3.2 Service Time

The second metric, average ST, is another critical metric for evaluating task coordination performance. ST is defined as the time between the origin sending the task offload and getting a response back – signifying that the task is finished. The benchmark is expressed

in seconds and indicates the system’s overall responsiveness and efficiency in handling tasks.

An optimal ST reflects a well-functioning system where tasks are offloaded to target nodes swiftly, leading to timely task completion and enhanced system performance. Evaluating ST helps identify the effectiveness of the three strategies in managing task coordination, particularly in their ability to minimize delays.



(a) ST in static environment.

(b) ST in dynamic environment.

Figure 5.6: ST in static and dynamic environments with ten devices.

The ST metric with ten nodes reveals several interesting observations about the performance of the Random, ACO, and Gossips strategies, illustrated in Figure 5.6. On the one hand, the Random strategy shows similarly good ST values as the more complex ACO approach. On the other hand, the former method has a more prominent outlier and a general tendency to higher values due to its random nature. That Random performs similarly well may be due to the definition of the metric: ST only considers those tasks for which the origin node has received a response, which indicates that the target successfully processed the task. Due to the way the strategy works, longer paths are built with larger cluster sizes, and the probability increases that the offload requests along these paths do not find any target nodes until the TTL is over. For this reason, it is expected that shorter paths are more likely to be returned, which thus has a shorter transmission time, which influences the ST. In short, the effect can be that Random gives better results than ACO, even though it can successfully offload fewer tasks. Therefore, we also include the Figure 5.7 of the GR, which provides insight into the number of tasks successfully returned versus the number of offloaded tasks. We can see that ACO performs better than Random at all cluster sizes despite similar STs. Furthermore, ACO has no path optimization, such as Random and Gossips, which can again reduce the response transmission time and, thus, the total ST.

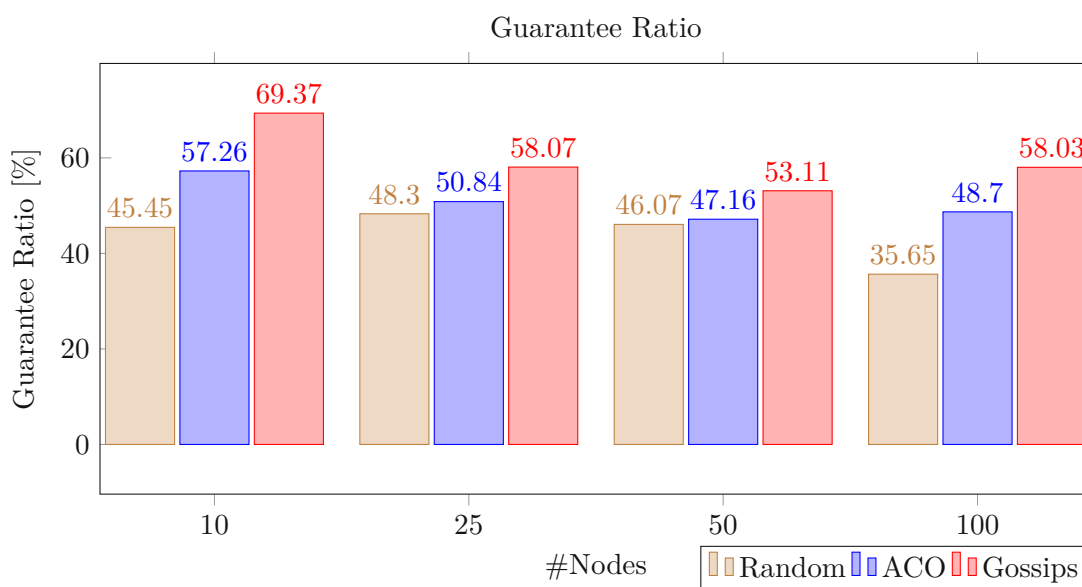
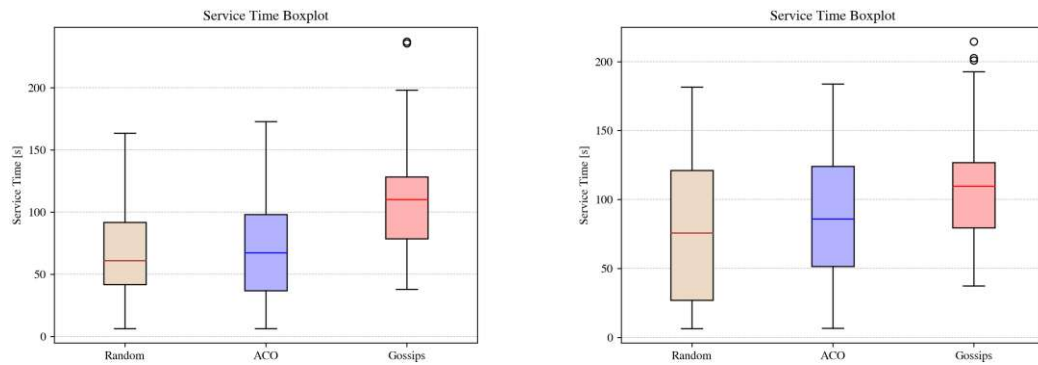


Figure 5.7: GR in static environment.

ACO demonstrates the most stable results, with the lowest spread in the box plot for both static and dynamic settings. This consistency indicates that ACO is more reliable in maintaining a stable ST across different tasks and network conditions. The Gossips performed the worst of the three strategies and has the highest ST. The high values have their origin in the 2-phase mode of operation. The first phase depends on the task TTL multiplied by a constant. Only after the phase is finished the target node is contacted. In the dynamic setting, Random and Gossips display higher spreads in ST compared to ACO. However, they manage to achieve shorter STs overall. It is worth mentioning that comparing the performance of these strategies in a dynamic environment can be challenging due to the varying number of device failures and recoveries in each test run. Each strategy has to deal with a different number of node failures and recoveries. In this case, Random has four recoveries, ACO has 12, and Gossips has 20. The more there are failures, the higher the probability that the system has less to do because more nodes are offline and waiting to recover.

Figure 5.8 presents that both Random and ACO display similar results in terms of spread and central tendency with 25 devices in the network, indicating that they manage to maintain comparable STs, even though Random achieves fewer tasks to be successfully offloaded. Furthermore, both probabilistic strategies do not display any outliers. This observation suggests that both Random and ACO have stable ST at a cluster size of 25 nodes. In contrast, the Gossips strategy exhibits a higher ST with outliers leaning towards the higher end of ST values. Additionally, it demonstrates a higher tendency for skewness toward higher STs. This skewness indicates that Gossips may be less reliable in maintaining a stable ST with a growing number of nodes.

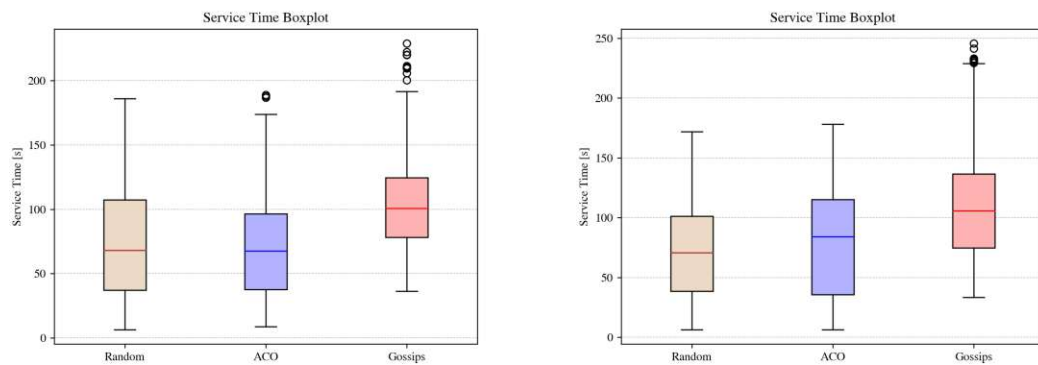


(a) ST in static environment.

(b) ST in dynamic environment.

Figure 5.8: ST in static and dynamic environments with 25 devices.

Skipping the analysis for 50 devices as it shows similar results, Table 5.9 examines the ST metric with 100 devices in the network. The evaluation of the three strategies provides the following insights: In the static environment, ACO exhibits a lower spread than the Random approach, signifying a more stable and consistent performance with a growing number of nodes. This stability is especially crucial when dealing with many devices, as it helps maintain an efficient and reliable system. We also assume that Random will increasingly show its weaknesses at larger cluster sizes, while ACO’s learning process will benefit the strategy. Gossips demonstrates more outliers than previously observed, both in static and dynamic settings. Gossips’ performance may become less consistent and less predictable as the number of devices increases, potentially impacting overall network efficiency. As with previous evaluations, Random and ACO show similar central tendencies in their STs.



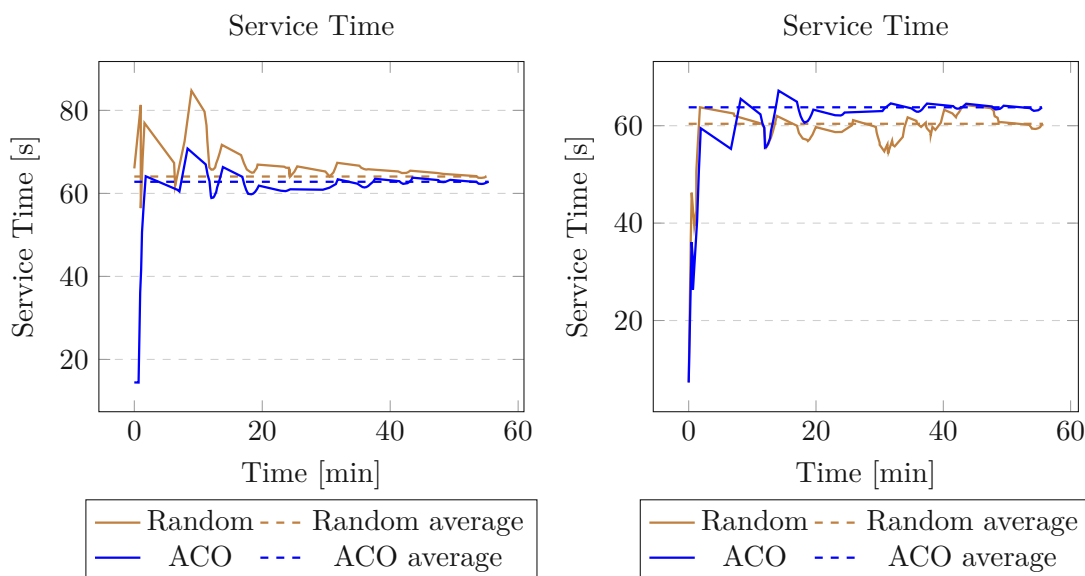
(a) ST in static environment.

(b) ST in dynamic environment.

Figure 5.9: ST in static and dynamic environments with 100 devices.

Since Random and ACO have similar ST values, we look closely at their behavior over time in the following sections. We will examine how the two strategies develop compared to their average values during a test run.

Analyzing the ST metric for Random and ACO strategies with ten nodes throughout the experiment duration provides valuable insights into their performance (Figure 5.10). Firstly, we notice that ACO's ST consistently remains below Random's ST in the static environment. It indicates the efficiency of the ACO strategy in managing tasks, as it consistently outperforms Random in this setting. In the dynamic environment, however, ACO's ST appears to be higher than Random's. This result suggests that the network topology and the number of devices failing and rejoining the network significantly impact the performance of these strategies in dynamic settings. ACO and Random show fluctuations around the average ST, whose cause may lie in the varied workload. ACO has a smaller amplitude than Random. A smaller amplitude indicates less significant deviations from the average ST, suggesting a more stable and reliable performance. Conversely, a larger amplitude demonstrates a less robust result, as in Random. Moreover, the function graphs reveal the task emitter frequencies, as the patterns in ST follow the task arrival rates. This observation further emphasizes the importance of understanding the relationship between task frequencies and the performance of the strategies.



(a) ST over time in static environment.

(b) ST over time in dynamic environment.

Figure 5.10: ST over time in static and dynamic environments with ten devices.

Figure 5.11 reveals that in the static environment, ACO and Random exhibit almost identical patterns in their STs, suggesting comparable performance levels. However, ACO's ST is slightly higher than Random's. Nonetheless, ACO can offload more tasks

simultaneously, indicating better task management capabilities. When examining the dynamic environment, the performance of both strategies is affected by the network topology and the number of devices failing and rejoining. In the ACO setting, over 56% of devices failed and rejoined; in the Random setting, this number was higher at 72%. This difference in device failure and rejoining rates results in a noticeable drop in the performance graphs for both strategies.

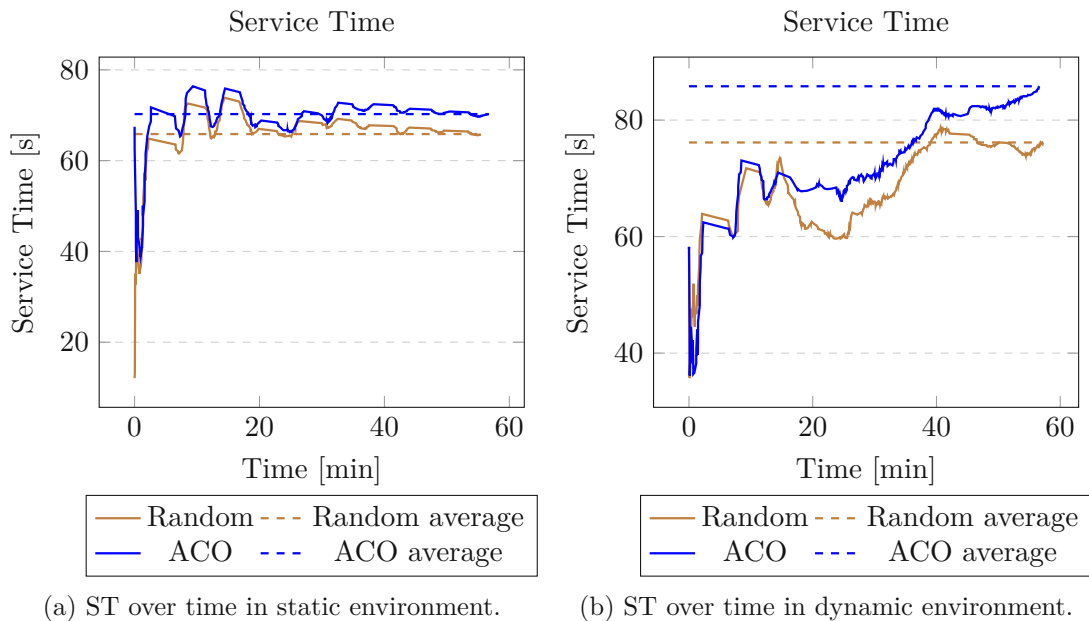
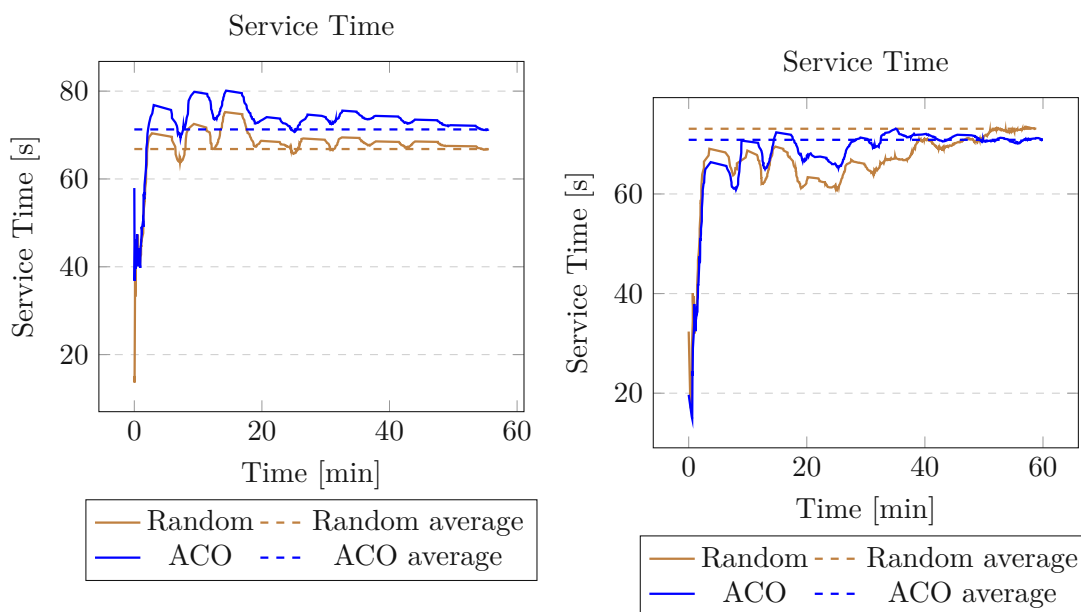


Figure 5.11: ST over time in static and dynamic environments with 25 devices.

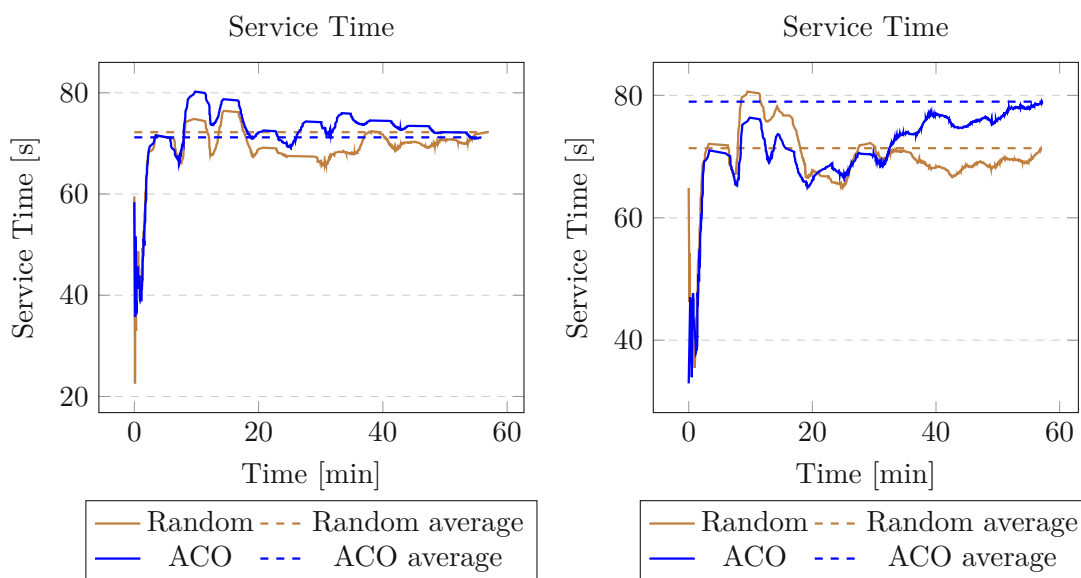
In static and dynamic environments, ACO and Random demonstrate similar patterns in their STs, suggesting comparable performance levels (Table 5.12). However, ACO has again a slightly higher ST than Random, as seen in the previous paragraph. The cause could be in the following three factors: 1.) the higher processing time required for ACO due to the calculation of the state transition rule, while Random selects a random path. 2.) Offload requests are often lost in the cluster when using Random, as it moves the task to any node until the TTL expires — only those tasks with a short path return, resulting in a reduced ST. 3.) Random benefits from path optimization, while ACO cannot do that due to pheromone deposition. In the dynamic environment, ACO outperforms Random towards the end of the experiment, with a decreasing ST compared to Random's increasing ST.



(a) ST over time in static environment.

(b) ST over time in dynamic environment.

Figure 5.12: ST over time in static and dynamic environments with 50 devices.



(a) ST over time in static environment.

(b) ST over time in dynamic environment.

Figure 5.13: ST over time in static and dynamic environments with 100 devices.

The ST observed in the 100 nodes experiment of Figure 5.13 is in a similar range as that of the 50 nodes experiment, indicating consistent behavior across the two different cluster

sizes. Towards the end of the experiment, ACO's ST appears to improve, gradually reducing the metric value as time progresses. This improvement suggests that the ACO strategy might adapt more effectively to the cluster's conditions, allowing for a better distribution of tasks and more efficient use of available resources. On the other hand, the ST for the Random strategy increases toward the end of the experiment, indicating a potential decrease in efficiency. The randomness of the strategy's approach may result in a less optimal distribution of tasks and resource allocation, especially as the network grows in size and complexity.

In summary, evaluating ST provides valuable insights into the performance of the Random, ACO, and Gossips strategies in task coordination. The analysis reveals that the performance of the Random and ACO strategies are similar in terms of ST, with ACO typically demonstrating a lower spread and a more stable result. The Gossips strategy consistently exhibits higher STs and greater skews towards higher values, attributed to its two-phase approach. The evaluation over time allowed for a more detailed understanding of the ST metric, revealing patterns in the performance of ACO and Random. ACO often shows a smaller amplitude in the fluctuations of ST compared to Random, indicating a more robust and stable performance. The comparison between the three strategies highlights some key lessons: The Random strategy, despite its simplicity, performs relatively well in the experiments, in the context of ST. However, tasks that get lost while traversing the graph are not considered, potentially leading to overestimating the Random strategy's performance. Furthermore, Random benefits from the path optimization technique. On the other hand, Gossips shows higher STs and a greater tendency towards outliers, which can be detrimental in real-world scenarios where fast task coordination is essential.

### 5.3.3 Hops Per Hit

This section focuses on the last metric, HPH, another essential factor in evaluating decentralized task coordination strategies. The benchmark refers to the number of hops or nodes traversed between the origin and target nodes during the task offloading process. A lower metric value indicates fewer messages were sent out. Moreover, the target node is relatively closer to the origin node, which can reduce service time because less latency is involved.

Assessing the average HPH metric allows us to understand the efficiency and scalability of the Random, ACO, and Gossips strategies in the context of message propagation and task coordination. A smaller number of HPH signifies faster communication between nodes, reducing communication overhead and shorter service time.

By evaluating the HPH metric, we can quickly identify the winner in this category. Figure 5.14 shows that ACO clearly outperforms the remaining two strategies. ACO finds the shortest path to a target node to offload the task. Compared to Random, ACO performs better by a factor of 2-3. As the number of nodes increases, ACO demonstrates robust scalability in static and dynamic environments. As expected, the strategy Random

performs the worst in static settings. The Gossips strategy underperforms in dynamic environments. Moreover, we can see that Random has similar values for the static and dynamic environments. The same applies to Gossips. However, the ACO algorithm copes much better with a static setting and struggles with dynamicity. The learned knowledge is lost due to the many device failures, and the pheromones must be redeposited. A noteworthy observation is that the ACO's performance with 25 nodes is slightly worse than with 50. It can be explained by the topology and connections between nodes, which influence the overall performance.

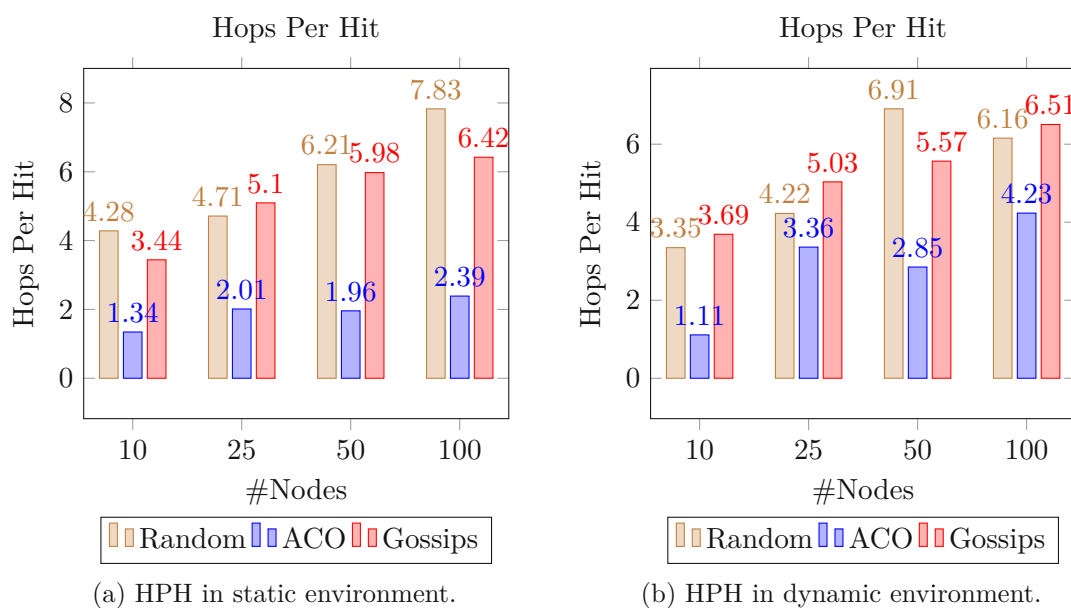
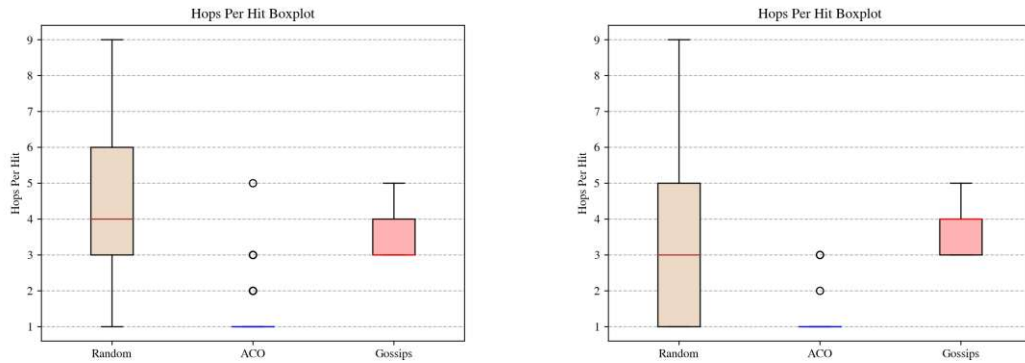


Figure 5.14: HPH in static and dynamic environments with ten devices.

In evaluating 10-node clusters using boxplots, we observe notable differences and trends among the three strategies. First, Random exhibits the highest HPH spread in both static and dynamic environments and the highest median in the static setting. We conclude that the performance of this approach can vary greatly and has no reliable task coordination mechanism. At ACO, on the other hand, we see a vastly different picture: The strategy demonstrates the best HPH with an almost negligible spread, except for a few outliers. The lack of fences in ACO's boxplot indicates a more consistent and stable performance, making it a promising candidate for decentralized task coordination. Gossips, however, displays the worst median in dynamic settings. Additionally, the algorithm has no lower fence, i.e., its lower fence equals the median.

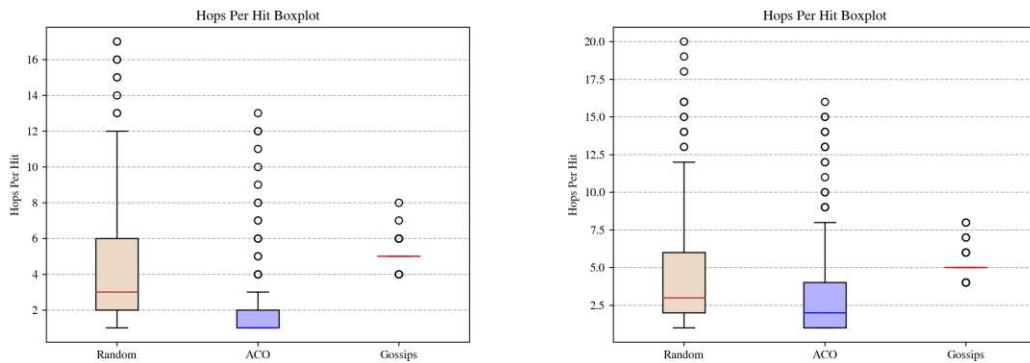


(a) HPH in static environment.

(b) HPH in dynamic environment.

Figure 5.15: HPH in static and dynamic environments with ten devices.

We can see a similar picture in the test runs with a cluster size of 25 nodes. Table 5.16 shows that ACO has by far the lowest HPH values, both in static and dynamic environments. However, the strategy shows a small spread and several outliers towards the upper fence. Nevertheless, 50% of all values are below 2 HPH. Gossips has the highest HPH and only a few outliers, but the median is the worst of the three approaches. With Random, we are beginning to see the effects of the arbitrary decisions: The algorithm shows strong outliers and tends to a higher HPH because, with a larger number of nodes, the risk of building a longer path without paying attention to the properties of the cluster increases.



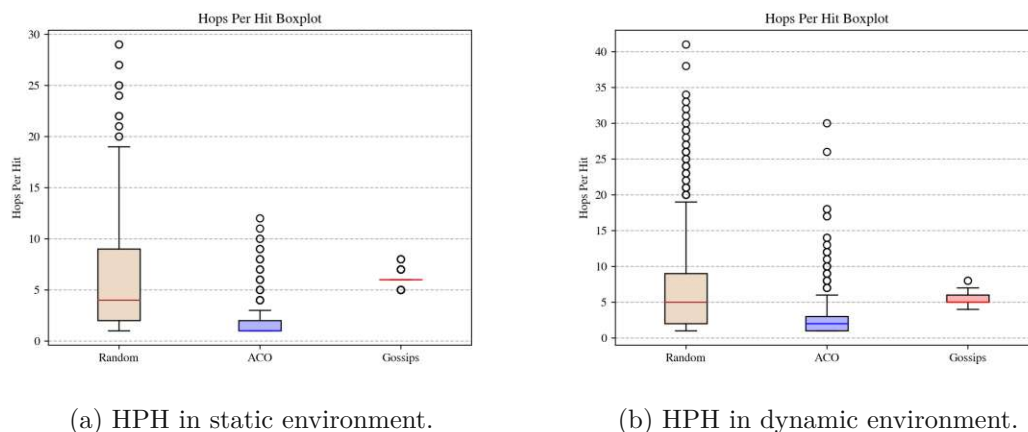
(a) HPH in static environment.

(b) HPH in dynamic environment.

Figure 5.16: HPH in static and dynamic environments with 25 devices.

The following Figure 5.17 shows how the effects of the observations are reinforced with 50 devices. Random performs increasingly worse as the cluster size grows. The HPH and the outliers grow. Although ACO also has more outliers, the spread remains very small,

and it finds the majority of target nodes below three hops. Gossip also performs strongly with only a few outliers and can cope equally well with and without device failures and rejoins.

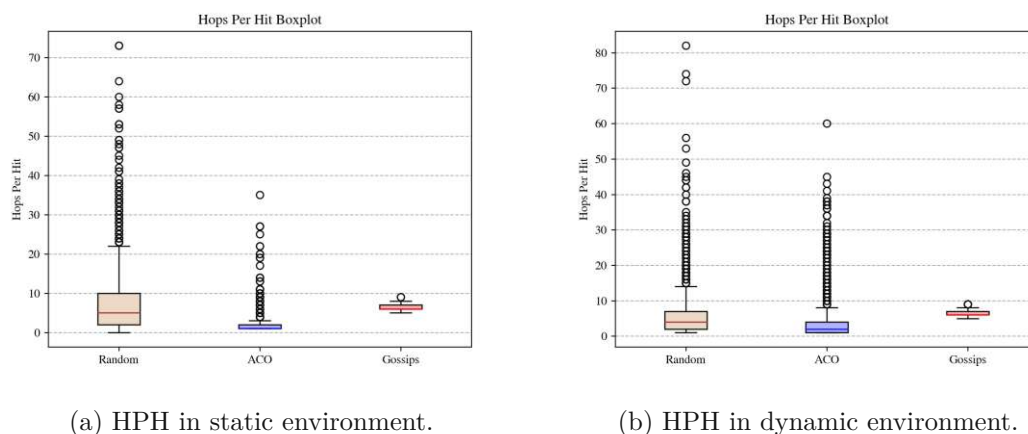


(a) HPH in static environment.

(b) HPH in dynamic environment.

Figure 5.17: HPH in static and dynamic environments with 50 devices.

Figure 5.18 summarizes the test run with 100 nodes. Like a cluster size of 50, Random is the worst algorithm in terms of HPH, while ACO is the best. Compared to the previous result, the two probabilistic strategies have more outliers. Random's highest value is even 75, meaning one-third of all nodes were traversed until the task offload request was accepted. Gossips again shows stable and consistent values in both environments.



(a) HPH in static environment.

(b) HPH in dynamic environment.

Figure 5.18: HPH in static and dynamic environments with 100 devices.

In summary, the ACO strategy has the lowest and, thus, best HPH values in all test runs, thanks to the deposition of pheromones. With increasing cluster size, ACO gets

## 5. EVALUATION

---

more outliers but still performs best. It would be interesting to know if the outliers are caused by the fact that the system needs more time to learn due to the larger number of nodes. I.e., the long paths (outliers) are formed at the beginning because not enough artificial ants have explored the network. Also, Gossips scores well in this category and shows solid stability and consistency in static and dynamic environments.

# Discussion

This chapter gives a summary overview of the performance and compares the different strategies, Random, ACO, and Gossips, to see in which application areas they would behave how. The chapter ends with a list of the limitations of this work, which are due to assumptions introduced at the beginning of the design phase.

The chapter is structured as follows: Section 6.1 provides the reader with a comprehensive comparison between the three strategies, including the communication metrics. Section 6.2 depicts how our system would behave in different application domains. We describe the limitations of this work in the last section 6.3.

## 6.1 Comparison of Strategies

In Chapter 5, we conducted an in-depth analysis of the single coordination metrics for Random, ACO, and Gossips strategies. Building on this foundation, we aim to examine the interrelation of these metrics, incorporating communication metrics in both environments as discussed in Zeqo's work [23]. By comparing the different strategies' performances in the seven metrics (LD, ST, HPH, AM, MPR, HMR, and GR), we strive to summarize the strengths and weaknesses of each algorithm. It helps us identify the most effective approach for different scenarios and potentially guides future research in optimizing these strategies.

To this end, we compute the average values for each metric and truncate them after two-comma digits to improve readability. The values are color encoded and signify how well the strategy ranks compared to the other approaches with the same cluster size. The winner in the metric and cluster size category appears in green, the second place gets yellow, and the last place is colored in red. For LD, weighting is based on whether the queue is neither too full nor too empty – strategies should target the occupation of 3/4 of the queue. We look at queues with a capacity of 15. For ST, HPH, AM, and MPR,

lower values are better. The lower the value, the better the performance of the whole system. For the last two metrics, HMR and GR, it is the other way around: the higher the strategies score in these categories, the better they perform.

Strategy	#Nodes	LD	ST	HPH	AM	MPR	HMR	GR
Random	10	4.46	64.06	4.28	713	5.40	45.45	45.45
	25	4.97	65.85	4.71	3215	7.29	48.29	48.29
	50	5.14	66.80	6.20	8865	11.40	46.07	46.07
	100	16.73	72.21	7.82	93818	31.50	35.75	35.65
ACO	10	5.03	62.79	1.34	300	2.37	57.26	57.26
	25	5.53	70.26	2.01	2309	4.75	50.84	50.84
	50	5.40	71.27	1.95	4664	5.75	47.15	47.15
	100	5.75	71.19	2.38	14243	8.93	48.69	48.69
Gossips	10	4.60	106.35	3.44	3900	33.42	69.36	69.36
	25	4.58	107.34	5.09	37400	114.01	58.07	58.07
	50	4.38	108.14	5.97	163549	296.12	53.11	53.11
	100	4.47	103.68	6.42	827354	757.37	58.02	58.02

Table 6.1: Comparison of strategies in static environment.

In Table 6.1, we can observe that ACO best distributes the tasks to the devices. Furthermore, the strategy requires the least hops to find a suitable target node for processing the offloaded task. Thanks to design decisions such as omitting multiple ants or iterations, as described in Section 4.3.2, ACO requires significantly fewer messages than the rest of the approaches. Thus, the strategy shows good scalability even as the number of nodes grows. Gossips, meanwhile, generates exponentially more messages, most of which are attributed to the first phase of the algorithm, where the network is flooded to find paths to possible target nodes. On the one hand, Gossips scales worse than ACO. On the other hand, the strategy has better reliability. It outperforms the other two approaches and processes significantly more tasks requested to be offloaded. That Random takes the first place in ST, and not ACO, could account for three reasons: 1.) Random’s simple mechanism of forwarding the task to a randomly chosen neighbor has the smallest computation overhead. 2.) Offload requests are often lost in the cluster when using Random, as it moves the task to any node until the TTL expires — only those tasks with a short path return, resulting in a reduced ST. 3.) Random uses path optimization, while ACO cannot profit from this mechanism.

In summary, ACO shows itself to be a promising candidate for task coordination in the static setting with room for improvement in the categories ST and GR. Gossips also proves to be a solid task coordination strategy with strength in HMR and GR, while ST, AM, and MPR suffer because of the naive Gossips protocol. An interesting finding is that ACO always performs better in LD and HPH. However, both strategies use the same scheme for quality: ACO in the form of pheromones and Gossips as sorting the results obtained from the first phase. Both take the path length between the origin and the

target and form the product with the target’s resource average and the average target’s queue occupation.

Next, we examine Table 6.2, which compares performance in a dynamic environment where devices can crash and recover. The strengths and weaknesses of each strategy remain the same: ACO finds a target node with the smallest number of hops and messages, while Gossips successfully offloads tasks the most often. It is noticeable here that the values of HMR and GR differ more than in static configuration. Apparently, nodes that are part of a response path crashed, which could explain the higher HMR values.

In general, we can conclude that IoT’s dynamic nature significantly impacts task coordination performance. We see a significant increase in the duration of an offload request and the number of messages generated. Implicitly, it also harms MPR.

Strategy	#Nodes	LD	ST	HPH	AM	MPR	HMR	GR
Random	10	7.35	60.39	3.34	1082	5.07	43.19	43.19
	25	9.47	76.15	4.22	8220	8.40	42.33	41.30
	50	8.62	73.09	6.90	20100	13.74	45.55	44.39
	100	7.16	71.37	6.15	30211	13.70	48.79	47.02
ACO	10	4.97	63.78	1.11	295	2.36	52.94	52.94
	25	11.48	85.80	3.36	8354	7.99	40.22	38.79
	50	6.30	70.87	2.85	6693	7.00	47.06	46.75
	100	8.97	78.97	4.23	32160	12.03	47.64	46.44
Gossips	10	3.69	91.27	3.69	3259	32.47	78.16	74.71
	25	5.10	106.92	5.03	43816	121.22	58.60	58.60
	50	5.92	104.56	5.56	196590	304.28	51.20	51.04
	100	5.88	110.33	6.50	849374	720.20	52.32	52.23

Table 6.2: Comparison of strategies in dynamic environment.

## 6.2 Use Cases

To better understand the practical implications of our findings, we can discuss real-world examples where the ACO and Gossips strategies may be employed, observing the expected behavior of each approach in these contexts.

For a real-world example where ACO may be suitable, we consider a remote monitoring system for wildlife conservation. In this scenario, IoT devices are deployed across vast areas to monitor and track the movement of endangered species, assess habitat conditions, and detect trend changes. Communication and contacting nodes can be expensive in such settings due to the long distances and limited resources. The limited infrastructure and unreliable connectivity make establishing and maintaining communication links challenging. These devices operate for several months or years, so ACO builds a better knowledge of the network topology with each offload request. It saves more hops and messages and increases the number of successfully offloaded tasks. Furthermore, an IoT

device does not have to offload a task as soon as it is overloaded but can process it locally, even if it takes longer. The processed data is often only analyzed later in such use cases. Tasks can be defined in the system that must be offloaded because of specific QoS (e.g., time-sensitive systems). Due to the self-learning cluster, ACO quickly finds a target node. The ACO algorithm would be a fitting choice with its efficient task distribution and minimal communication overhead. By employing ACO, conservationists can expect a more resource-efficient system, enabling them to collect valuable data with reduced communication costs.

As for an example where Gossips may be more appropriate, consider a distributed sensor network in a smart city infrastructure. IoT devices are interconnected in this setting to manage various aspects such as traffic control, air quality monitoring, and public safety. In this scenario, it is crucial for nodes to help each other and ensure the system is reliable. Additionally, communication costs are relatively low due to the high availability of networking infrastructure in urban environments. The strategy would be well-suited for this context with its strength in offloading tasks and achieving high reliability. Furthermore, the cloud can be a part of the system design. IoT devices support themselves in the cluster but also have the possibility to contact strong cloud centers for non-time-sensitive tasks. The advantages are twofold: 1.) The cloud is relieved because devices exchange tasks with each other and thus reduce latency and bandwidth. 2.) At the same time, IoT devices have access to practically infinite resources from the cloud if required. By implementing Gossips in an intelligent city sensor network, city planners and administrators can expect a more dependable system that effectively handles diverse tasks and maintains a high level of performance, ensuring the smooth operation of vital city services.

In summary, our research reveals that ACO and Gossips are applicable approaches for task coordination in IoT settings, each with unique advantages depending on the context, but also with their drawbacks. ACO excels in environments where communication efficiency is paramount, minimizing hops and messages sent across the network. At the same time, Gossips is better suited for settings where reliability and successful task offloading are the main priorities to relieve locally overloaded nodes. The dynamic nature of heterogeneous IoT devices in a cluster significantly impacts task coordination performance, so understanding these strategies' behavior and applicability in real-world scenarios is crucial.

This study serves as a foundation for future work to improve these decentralized algorithms and further explore their potential in various IoT use cases. By examining their performance in static and dynamic environments, we can provide valuable insights into their applicability and potential for further optimization, ultimately paving the way for more efficient and reliable task coordination in the ever-growing IoT landscape.

## 6.3 Limitations

During the design phase of our thesis, it was required to draw boundaries and make certain assumptions, which inevitably led to some limitations. We encountered additional constraints impacting our work as we progressed through the design and implementation phases. Furthermore, the evaluation process shed light on new limitations that emerged from the interaction of our strategies with the testing environments. These limitations reflect the scope of our current research and highlight areas where further investigation and improvement are needed. By acknowledging these constraints, we can provide a more comprehensive understanding of our work and lay the foundation for future research efforts in the field of decentralized task coordination in distributed IoT systems.

First, the evaluated experiments have shown that ACO shows stronger deviations in the dynamic setting than in the static one. Furthermore, HPH is noticeably higher when device failures are present. On the one hand, this behavior cannot be avoided entirely because the static setting is always easier for coordination. On the other hand, performance can be improved by handling the pheromone tables for crashed devices differently. Currently, it is the case that all pheromones are lost for the device that is no longer in the cluster. Consequently, all collected information about the device and the path with other nodes is also lost. Instead of deleting all pheromones, a relative amount could be subtracted for each failure to mark more frequent device outages as unwanted, but also to avoid losing all pheromones when the device comes back online.

Second, due to our decision to offload all newly arriving tasks immediately when the queue capacity is reached, the probability is very high that offloaded tasks will be processed earlier than tasks processed by the device itself when the capacity was still sufficient. This approach implies that all tasks must be independent of each other. Furthermore, they cannot be tasks containing real-time information because new information can be overwritten. These assumptions offer limitations for certain IoT use cases. For this reason, this queue behavior should be extended in future work.

Third, our system does not have security mechanisms to guard against typical threats in distributed systems, such as Sybil or Sinkhole attacks. Although we recognize that incentives, security, and trust mechanisms are essential for a successful task coordination strategy, this work does not address these issues. For the purposes of this paper, we have assumed that incentive mechanisms are already in place and that our system is run in a secure environment.

Fourth, the ACO strategy, based on AS, is susceptible to the parameters that control the impact on global and local knowledge. Parameter tuning tests can significantly improve the performance of ACO and can be adapted depending on the use case, which is not mentioned in our thesis. This work aimed not to show parameter-optimized results but to use general values to present a possible baseline for different application purposes.

Fifth, a notable aspect of our test design is the balance between efficiency and comprehensiveness. Given the time-consuming nature of the tests, we opted to perform a single test

## 6. DISCUSSION

---

run for each experiment combination rather than conducting multiple runs and averaging the results. While this approach made the testing process more manageable, extensive testing would reduce the occurrence of outliers in the results. The added precision might facilitate more meaningful comparisons between the strategies, particularly in dynamic environments. Consequently, further research could benefit from extensive testing.

Finally, in our system, several clusters may be formed. Clustering in itself is not negative and did not occur in the experiments. However, this behavior is unexplored, and the performance impact of devices splitting into multiple clusters is unknown. Since clustering is a separate topic, our work did not address this characteristic feature.

## Conclusion

In this work, we presented three decentralized task coordination strategies, Random, ACO, and Gossips, which handle the distribution of tasks from overloaded devices in a distributed IoT system. We evaluated and compared the performance of the three approaches by conducting experiments with a different number of IoT devices, which acted statically or dynamically with failures and re-entries. We concluded that the ACO algorithm we developed performed better than the baseline, Random, and is a topic for future work in decentralized task coordination.

The need for decentralized task coordination originates from the strong dependency of IoT devices on the cloud. Since all storage and data processing happens in cloud centers, bottlenecks and limitations arise for distributed IoT systems. Locally available resources are not considered, while latency and bandwidth in the centralized IoT model grow and cannot scale over time. Instead, a decentralized model is necessary in which IoT devices communicate and collaborate within clusters. These so-called swarms adapt to IoT devices' heterogeneous and dynamic nature and are self-learning, self-managing, and self-adapting during runtime. Thus, new challenges are emerging to enable this new shift from a centralized to a decentralized architecture. One of them is how devices can coordinate tasks for themselves. A cluster consists of devices equipped with different resources and capacities. The goal is that resource-constrained devices that are overloaded can find more powerful devices in the cluster and offload their workload to them.

This thesis looked at decentralized task coordination in distributed IoT systems. The devices, which all have different queue sizes, periodically produce tasks that need to be processed. If the queue is over its capacity, the device tries to find other cluster members with resources available to offload tasks. This thesis aimed to design and implement task coordination methods based on requirements for distributed IoT systems, including heterogeneity, processing and communication overhead, scalability, and dynamic environment. To this end, three different strategies were employed. 1.) The baseline algorithm randomly decides which node should be contacted next for offloading. 2.) The

ACO approach, which spreads pheromones along the cluster that indicate how attractive the paths to the offload nodes are. We introduced single forward ants carrying the offload request and backward ants depositing the pheromones. 3.) The Gossips strategy, which tries to ask all nodes simultaneously if they can process the task. Based on these three approaches, the following objectives were addressed in this work: 1.) How to efficiently coordinate tasks in a dynamic, resource-constrained, and decentralized IoT setting, where nodes can fail, leave, and rejoin the network at any time? 2.) How does the decentralized Ant Colony Optimization-based algorithm impact the task coordination performance compared to the Random and Gossips-based approaches? 3.) How does the dynamic nature of heterogeneous IoT devices in a cluster impact the task coordination performance of the decentralized algorithms? Our central work core was to take AS as the basis for our ACO algorithm, adapt it to decentralized task coordination, and see how well it performed despite limited local knowledge and without multiple iterations or ants.

Furthermore, we conducted several experiments to evaluate our three strategies according to different metrics. These included LD, which gives insight into the average queue occupation; ST, which shows how long it takes to offload a task; and HPH, which indicates after how many hops one has found a matching device. The experiments were run with 10, 25, 50, and 100 devices in a static and dynamic environment where devices can fail and rejoin. The results showed that ACO performed better than the other strategies in LD. However, all three approaches have room to improve in this metric, especially at smaller queue capacities. We also observed that Random and ACO have similar ST, but at the same time, ACO successfully offloads a task more often. The Gossips strategy has the longest ST because of its 2-phase design. Finally, we could assess that ACO has the strongest HPH values in both the static and dynamic environments, outperforming the other approaches.

In conclusion, ACO, which performs better than the baseline Random, is shown to be a functional approach for decentralized task coordination. It provides a reasonable basis for further work in this direction, summarized below.

### 7.1 Future Work

This thesis gave a solid overview of the three decentralized task coordination strategies and their performance in the different metrics and experiments. Building on the results of our work and the identified limitations described in Section 6.3, we can use this knowledge to shape and extend the strategies, their design decisions, and the test design more sophisticatedly. This section proposes future research directions that can positively impact coordination performance. We also make additions to our design choices and present new ideas.

First, exploring the graph traversal behavior in our thesis could be further enhanced by incorporating path optimization for ACO. Currently, the lack of path optimization prevents accurate pheromone deposit calculations, potentially affecting coordination efficiency. Additionally, the possibility of reverting to previous states upon detecting

cycles should be considered for all strategies. In the current implementation, offload requests caught in cycles are dropped, leading to the loss of task processing opportunities.

Second, our test setup could benefit from additional experiments and improvements. Exhaustive parameter testing for ACO under diverse application settings could yield a better understanding of which parameters perform optimally in specific scenarios, such as real-time sensitive applications or larger cluster sizes. Furthermore, expanding tests to include more than 100 nodes would provide insights into the scalability of the proposed strategies. Real-world task workloads and different emitter times should also be investigated to examine the system's performance under various application requirements. Lastly, replacing simulated resources with VMs of different configurations could offer more realistic testing conditions and help assess the impact of these configurations on performance.

Third, handling tasks can be improved and extended by splitting tasks into subtasks. It facilitates finer-grained control over task distribution and processing. Therefore, this approach could lead to more efficient resource utilization and improved load balancing across the system. Another aspect to consider is the introduction of dependent tasks, where the order of processing and offloading tasks is crucial for the correct functioning of the system. Incorporating support for such tasks would broaden the applicability of our strategies and allow for a more comprehensive evaluation of their efficiency in diverse scenarios. Another aspect of extending our system is that the current implementation only supports stateless tasks, limiting the scope of our strategies. By introducing input for tasks, we could enable support for a broader range of applications and use cases. Extending our strategies to handle tasks with input would provide a more versatile and robust solution for decentralized task coordination in IoT systems.

Fourth, the deposit of pheromones can be optimized by refining the initialization process when two devices are connected. Instead of depositing a constant value when adjacent nodes support the same task type, more sophisticated rules could be devised. Additionally, enhancing the re-initialization process for failed devices could help retain the knowledge accumulated over time when a node rejoins the network. We assume that the introduction of these two aspects can have a significant impact on ACO performance in dynamic environments.

Fifth, the pheromone calculation could also be improved by accounting for factors such as energy consumption or device proximity, which may influence a target node's suitability. Moreover, the reliance on the number of hops as a distance metric encoded in the pheromones may not accurately reflect the actual communication cost or latency in some cases, such as when nodes are connected through high-latency links.

Lastly, a promising avenue for future work involves the development of a hybrid solution that combines the strengths of both ACO and Gossips strategies. By leveraging the advantages of ACO, such as its superior target node identification, reduced message generation, and higher scalability, we could improve the overall efficiency of the task coordination process in distributed IoT systems. On the other hand, we observed that

## 7. CONCLUSION

---

Gossips performs better in dynamic settings, primarily due to its deterministic search space. By incorporating this aspect of Gossips into the hybrid solution, we could enhance the adaptability of our strategies, enabling them to better cope with changing network conditions and resource availability.

# List of Figures

1.1	Flow chart showing the various stages performed during the master thesis.	3
4.1	System context diagram of the framework. . . . .	25
4.2	Container diagram of the framework. . . . .	25
4.3	Component diagram of the framework. . . . .	26
4.4	Structure of the queue. . . . .	29
4.5	Path optimization illustration. . . . .	33
4.6	Concept of forward and backward ants. . . . .	36
5.1	Arrival rates for different task types. . . . .	53
5.2	LD in static (1 <sup>st</sup> row) and dynamic (2 <sup>nd</sup> row) environments with ten devices of different queue capacities. . . . .	57
5.3	LD in static (1 <sup>st</sup> row) and dynamic (2 <sup>nd</sup> row) environments with 25 devices of different queue capacities. . . . .	58
5.4	LD in static (1 <sup>st</sup> row) and dynamic (2 <sup>nd</sup> row) environments with 50 devices of different queue capacities. . . . .	58
5.5	LD in static (1 <sup>st</sup> row) and dynamic (2 <sup>nd</sup> row) environments with 100 devices of different queue capacities. . . . .	59
5.6	ST in static and dynamic environments with ten devices. . . . .	60
5.7	GR in static environment. . . . .	61
5.8	ST in static and dynamic environments with 25 devices. . . . .	62
5.9	ST in static and dynamic environments with 100 devices. . . . .	62
5.10	ST over time in static and dynamic environments with ten devices. . . . .	63
5.11	ST over time in static and dynamic environments with 25 devices. . . . .	64
5.12	ST over time in static and dynamic environments with 50 devices. . . . .	65
5.13	ST over time in static and dynamic environments with 100 devices. . . . .	65
5.14	HPH in static and dynamic environments with ten devices. . . . .	67
5.15	HPH in static and dynamic environments with ten devices. . . . .	68
5.16	HPH in static and dynamic environments with 25 devices. . . . .	68
5.17	HPH in static and dynamic environments with 50 devices. . . . .	69
5.18	HPH in static and dynamic environments with 100 devices. . . . .	69



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar  
The approved original version of this thesis is available in print at TU Wien Bibliothek.

# List of Tables

4.1	Tech stack of the framework. . . . .	26
4.2	Device's and tasks' properties. . . . .	28
4.3	ACO strategy's parameters. . . . .	40
4.4	Gossips strategy's parameters. . . . .	44
5.1	Device types and their characteristics. . . . .	51
5.2	Number of nodes and minimum connections per device type. . . . .	52
5.3	Device distribution in different cluster sizes. . . . .	52
5.4	Framework's parameter values used for the experiments. . . . .	54
5.5	Server hardware specification. . . . .	55
5.6	VM hardware specifications. . . . .	55
6.1	Comparison of strategies in static environment. . . . .	72
6.2	Comparison of strategies in dynamic environment. . . . .	73



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar  
The approved original version of this thesis is available in print at TU Wien Bibliothek.

# List of Algorithms

2.1	AS in pseudo-code. . . . .	11
4.1	The decentralized ACO strategy in pseudo-code . . . . .	37
4.2	The decentralized Gossips strategy in pseudo-code . . . . .	42



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar  
The approved original version of this thesis is available in print at TU Wien Bibliothek.

# Glossary

**ACO** Ant Colony Optimization based Task Coordination Strategy. 7



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar  
The approved original version of this thesis is available in print at TU Wien Bibliothek.

# Acronyms

- ACS** Ant Colony System. 17, 18
- AM** Amount of Messages. 50, 71, 72
- AS** Ant System. 10, 11, 18, 28, 33, 34, 36, 37, 39, 41, 85
- CCC** Cognitive Computing Continuum. 15, 16
- CPACO** Collection Path Ant Colony Optimization. 20
- FIFO** First In – First Out. 29
- FOHA** Forward Optimal Heuristic Algorithm. 20
- GR** Guarantee Ratio. 50, 60, 61, 71–73, 81
- GSA** Global Search Algorithm. 20
- HMR** Hit Miss Ratio. 50, 71–73
- HPH** Hops per Hit. 50, 55, 66–69, 71, 72, 78, 81
- IDC** International Data Corporation. 1
- JSON** JavaScript Object Notation. 30, 45
- LD** Load Distribution. 49, 55–59, 71, 72, 78, 81
- MAS** Multi-Agent Systems. 20
- MPR** Messages per Request. 50, 71–73
- ND** Node Discovery. 30, 37, 45
- PSO** Particle Swarm Optimization. 20, 21

- QoS** Quality of Service. 15, 16, 19–21, 74
- SA** Self-Actualization. 30, 45–47
- SACO** Smart Ant Colony Optimization. 21
- SI** Swarm Intelligence. 22
- ST** Service Time. 50, 55, 59–66, 71, 72, 78, 81
- TCP** Transmission Control Protocol. 30, 45
- TSP** Traveling Salesman Problem. 17, 18
- TTL** Time-To-Live. 18, 27, 28, 32, 35, 43, 44, 47, 50, 60, 61, 64, 72
- VM** Virtual Machine. 4, 19, 20, 27, 49, 51, 54, 55, 79, 83

# Bibliography

- [1] K. H. N. Bui and J. J. Jung. Aco-based dynamic decision making for connected vehicles in iot system. *IEEE Transactions on Industrial Informatics*, 15:5648–5655, 10 2019. ISSN 19410050. doi: 10.1109/TII.2019.2906886.
- [2] J. Calle, J. Rivero, D. Cuadra, and P. Isasi. Extending aco for fast path search in huge graphs and social networks. *Expert Systems with Applications*, 86:292–306, 11 2017. ISSN 0957-4174. doi: 10.1016/J.ESWA.2017.05.066.
- [3] G. Di Caro and M. Dorigo. Antnet: Distributed stigmergetic control for communications networks. *Journal of Artificial Intelligence Research*, 9:317–365, 1998.
- [4] M. Dorigo. Optimization, learning and natural algorithms. *Ph.D. Thesis, Politecnico di Milano*, 1992. URL <https://cir.nii.ac.jp/crid/1573950400977139328>.
- [5] M. Dorigo and L. Gambardella. Ant colony system: a cooperative learning approach to the traveling salesman problem. *IEEE Transactions on Evolutionary Computation*, 1(1):53–66, 1997. doi: 10.1109/4235.585892.
- [6] M. Dorigo, V. Maniezzo, and A. Coloni. Ant system: Optimization by a colony of cooperating agents. *IEEE transactions on systems, man, and cybernetics - part b. IEEE transactions on systems, man, and cybernetics. Part B, Cybernetics : a publication of the IEEE Systems, Man, and Cybernetics Society*, 26:29–41, 02 1996. doi: 10.1109/3477.484436.
- [7] J. Fan, X. Wei, T. Wang, T. Lan, and S. Subramaniam. Deadline-aware task scheduling in a tiered iot infrastructure. *2017 IEEE Global Communications Conference, GLOBECOM 2017 - Proceedings*, 2018-January:1–7, 7 2017. doi: 10.1109/GLOCOM.2017.8255037.
- [8] K. S. A. Fathima and K. Sindhanaiselvan. Ant colony optimization based routing in wireless sensor networks. *International Journal of Advanced Networking and Applications*, 4(4):1686, 2013.
- [9] A. J. Ferrer, S. Becker, F. Schmidt, L. Thamsen, and O. Kao. Towards a cognitive compute continuum: An architecture for ad-hoc self-managed swarms. *Proceedings - 21st IEEE/ACM International Symposium on Cluster, Cloud and Internet Computing, CCGrid 2021*, pages 634–641, 5 2021. doi: 10.1109/CCGRID51090.2021.00076.

- [10] J. Hojlo. Future of industry ecosystems: Shared insights & data, 1 2021. URL <https://blogs.idc.com/2021/01/06/future-of-industry-ecosystems-shared-data-and-insights/>.  
<https://blogs.idc.com/2021/01/06/future-of-industry-ecosystems-shared-d>  
last visited: 06.04.2023.
- [11] M. K. Hussein and M. H. Mousa. Efficient task offloading for iot-based applications in fog computing using ant colony optimization. *IEEE Access*, 8:37191–37201, 2020. ISSN 21693536. doi: 10.1109/ACCESS.2020.2975741.
- [12] A. Kishor and C. Chakarbarty. Task offloading in fog computing for using smart ant colony optimization. *Wireless Personal Communications*, pages 1–22, 7 2021. ISSN 1572834X. doi: 10.1007/S11277-021-08714-7/FIGURES/8. URL <https://link.springer.com/article/10.1007/s11277-021-08714-7>.
- [13] A. Kishor and C. Chakarbarty. Task offloading in fog computing for using smart ant colony optimization. *Wireless Personal Communications*, 127:1683–1704, 11 2022. ISSN 1572834X. doi: 10.1007/S11277-021-08714-7/FIGURES/8. URL <https://link.springer.com/article/10.1007/s11277-021-08714-7>.
- [14] S. Kumar, V. Kumar-Solanki, S. K. Choudhary, A. Selamat, and R. González-Crespo. Comparative study on ant colony optimization (aco) and k-means clustering approaches for jobs scheduling and energy optimization model in internet of things (iot). *International Journal of Interactive Multimedia and Artificial Intelligence*, 6:107, 2020. ISSN 1989-1660. doi: 10.9781/IJIMAI.2020.01.003. URL <https://reunir.unir.net/handle/123456789/12710>.
- [15] J. Manyika and M. Chui. By 2025, internet of things applications could have \$11 trillion impact, 7 2015. URL <https://www.mckinsey.com/mgi/overview/in-the-news/by-2025-internet-of-things-applications-could-have-11-trillion-impact>.  
<https://www.mckinsey.com/mgi/overview/in-the-news/by-2025-internet-of-things-applications-could-have-11-trillion-impact>,  
last visited: 06.04.2023.
- [16] E. Michlmayr. Ant algorithms for search in unstructured peer-to-peer networks. *ICDEW 2006 - Proceedings of the 22nd International Conference on Data Engineering Workshops*, 2006. doi: 10.1109/ICDEW.2006.29.
- [17] E. Michlmayr. Specification of the semant algorithm. *Tech. Rep.*, 2006.
- [18] T. C. Müller. MI-based power consumption prediction models for edge devices. Master’s thesis, TU Wien, 2023. URL <https://repositum.tuwien.at/handle/20.500.12708/175670>.
- [19] C. Tsigkanos, S. Nastic, and S. Dustdar. Towards resilient internet of things: Vision, challenges, and research roadmap. *Proceedings - International Conference on*

*Distributed Computing Systems*, 2019-July:1754–1764, 7 2019. doi: 10.1109/ICDCS.2019.00174.

- [20] M. Villari, M. Fazio, S. Dustdar, O. Rana, and R. Ranjan. Osmotic computing: A new paradigm for edge/cloud integration. *IEEE Cloud Computing*, 3:76–83, 11 2016. ISSN 23256095. doi: 10.1109/MCC.2016.124.
- [21] L. Wang, Z. Wang, S. Hu, and L. Liu. Ant colony optimization for task allocation in multi-agent systems. *China Communications*, 10:125–132, 3 2013. ISSN 16735447. doi: 10.1109/CC.2013.6488841.
- [22] A. Zannou, A. Boulaaam, and E. H. Nfaoui. A task allocation in iot using ant colony optimization. *Proceedings - 2019 International Conference on Intelligent Systems and Advanced Computing Sciences, ISACS 2019*, 12 2019. doi: 10.1109/ISACS48493.2019.9068889.
- [23] G. Zeqo. Decentralized communication in heterogeneous iot clusters. Master’s thesis, TU Wien, 2023.



Die approbierte gedruckte Originalversion dieser Diplomarbeit ist an der TU Wien Bibliothek verfügbar  
The approved original version of this thesis is available in print at TU Wien Bibliothek.

# Appendix

## Gitlab Repositories

- Framework  
<https://git.dsg.tuwien.ac.at/master-thesis/rain-cloud-system>
- Monitoring System  
<https://git.dsg.tuwien.ac.at/master-thesis/monitoring>
- Diagrams  
<https://git.dsg.tuwien.ac.at/master-thesis/documentation/-/tree/master/evaluation>