www.openswarm.eu

Call: HORIZON-CL4-2022-DATA-01

Type of action: RIA

Grant agreement: 101093046

Deliverable 2.5: Performance of the Coaty Framework Running on a TSCH-based Constrained Network

Work Package 2: Orchestration of Collaborative Smart Nodes

Task 2.4: Constrained Coaty

Task Lead: SIG

Task Contributor: SIA, ING, SIP

WP Lead: ADI

This project has received funding from the European Union's Horizon Europe Framework Programme under Grant Agreement No. 101093046.

Funded by the European Union

Document information	
Author(s)	Markus Sauer, José Quevedo, Jochen Nickles
Reviewers	Micael Couceiro
Submission date	31-Oct-2024
Due date	31-Oct-2024
Type	R
Dissemination level	PU

Document history			
Date	Version	Author(s)	Comments
30-Jun-2024	01	Jochen Nickles	preview
31-Oct-2024	02	Jochen Nickles	Deliverable

Table of contents

Executive Summary

This document constitutes Deliverable D2.5 of the OpenSwarm project, focusing on the evaluation of the Coaty framework in constrained network environments. Executed under Task T2.4, this effort aimed to adapt low-power wireless communication technologies to support distributed operations essential for the Coaty framework. The task commenced with a detailed examination of application overlays alongside their associated trade-offs, enhancing our understanding of Coaty's operation in constrained settings. This led to a critical evaluation of existing Coaty features against the foundational requirements outlined in WP1. The insights gained prompted the development of a specialised iteration of Coaty, termed "constrained Coaty", which is tailored for constrained networks. This new version underwent rigorous testing in various test setups to ensure its functionality and robustness. The successful completion of this task not only validates the new Coaty framework, but also sets the stage for its integration into the proof-of-concept (PoC) deployments planned in WP6, thereby enhancing collaborative communication capabilities within the OpenSwarm project.

Introduction

Recent advancements in communication, computation, and sensing, coupled with the ongoing progress in artificial intelligence, have positioned data at the centre of all technological developments. To unlock the full potential of these integrated technologies, ensuring seamless data exchange, synchronisation, and consistency across distributed collaborative nodes is imperative. Application overlays, serving as distributed data-centric frameworks, are a cornerstone for enhancing the efficiency of future industrial operations by facilitating effective data utilisation.

In this context, OpenSwarm aims to integrate and enhance existing communication protocols with robust peer-to-peer communication and port coordination mechanisms. Specifically, the project focuses on adapting Coaty, a lightweight open-source framework designed for collaborative IoT applications, to operate efficiently within constrained network environments.

Task T2.4, a core component of WP2's development activities, was established with the following objectives: (i) identify the optimal low-power wireless communication mechanisms to support distributed operations as defined by the Coaty framework; (ii) analyse the trade-offs associated with the use of application overlays; (iii) develop and release a "constrained Coaty" specification, along its implementation; and (iv) benchmark the performance of this new Coaty version in various test environments.

The remainder of the document is organized as follows:

- Chapter 2: Provides a detailed description of the methodology that guided the work conducted in this task.
- Chapter 3: Analyses the state-of-the-art in Application Overlay Networks
- Chapter 4: Details the design and implementation of the evolved Coaty tailored for constrained networks and specific OpenSwarm requirements.
- Chapter 5: Evaluates the novel Coaty in different test setups.
- Chapter 6: Provides a summary and conclusion of the findings.

Key Performance Indicators (KPIs)

With the introduction of the new constrained Coaty release, we aim to extend the battery lifetime of constrained Coaty-enabled devices. With the significantly smaller footprint from communication and computational perspective of the new release, we expect a significantly reduced power consumption. In addition, the new way of using datacentric communication like with constrained Coaty allows for a fine-grained control of communication performed to further support energy efficiency. Thus, it is expected that the KPI of more than two years of battery autonomy can be achieved. The validation of this KPI (as defined for T2.4 in Table 3.2 of DoA) will be further conducted during the realization of real-world tasks in the PoCs to assess these values.

Technology Readiness Level (TRL)

This section outlines the developments in the Technology Readiness Level (TRL) for the Constrained Coaty Framework, specifically within the scope of T2.4. Our work has advanced the Constrained Coaty from TRL2, focusing on technological principles, to TRL4, which involves lab-based validation using a PoC prototype in a concrete use case. The progression to TRL5, demonstrating the framework's functionality in relevant environments, is anticipated to occur through the execution of Coaty-enabled PoCs in WP6.

List of Publications

- J. Quevedo, M. Sauer, J. Nickles, **"Data-Centric Collaborative Distributed Systems: A System Engineering Perspective"**, is in preparation and will be submitted to IEEE Communication Surveys and Tutorials.

Methodology

The methodology adopted in this task is systematic and iterative, ensuring a thorough exploration and enhancement of the Coaty framework within constrained network environments. The approach is depicted in Figure 1 and comprises several sequential steps:

1. **State-of-the-Art Review:** Initially, we conducted a comprehensive review of the current technological landscape surrounding application overlay networks. This review helped us understand the core principles and latest advancements in this field, setting the stage for the subsequent adaptation of Coaty.
2. **Assessment Against OpenSwarm Requirements:** Following the review, we analysed how existing features of the Coaty framework align with the specific requirements outlined in WP1 of OpenSwarm. This assessment was crucial to identify gaps and opportunities for enhancement tailored to support distributed operations in constrained networks.
3. **Development of Constrained Coaty:** With a clear understanding of the necessary adaptations, we developed an initial release of the Coaty framework designed for constrained networks. This version incorporated modifications aimed at optimizing performance and resource utilization in line with OpenSwarm's objectives.
4. **Validation in Test Setups:** The modified Coaty framework was then rigorously tested in various setups that reflect the real-world conditions of OpenSwarm's POC4 use-cases. These tests were instrumental in validating the functionality and feasibility of using Coaty under constrained conditions.

5. **Deployment in PoCs and further development:** Insights gained from these tests will be leveraged for further refinement of Coaty. This ongoing development cycle will continue throughout WP6, where Coaty's integration and performance will be further assessed across different proof-of-concept implementations.

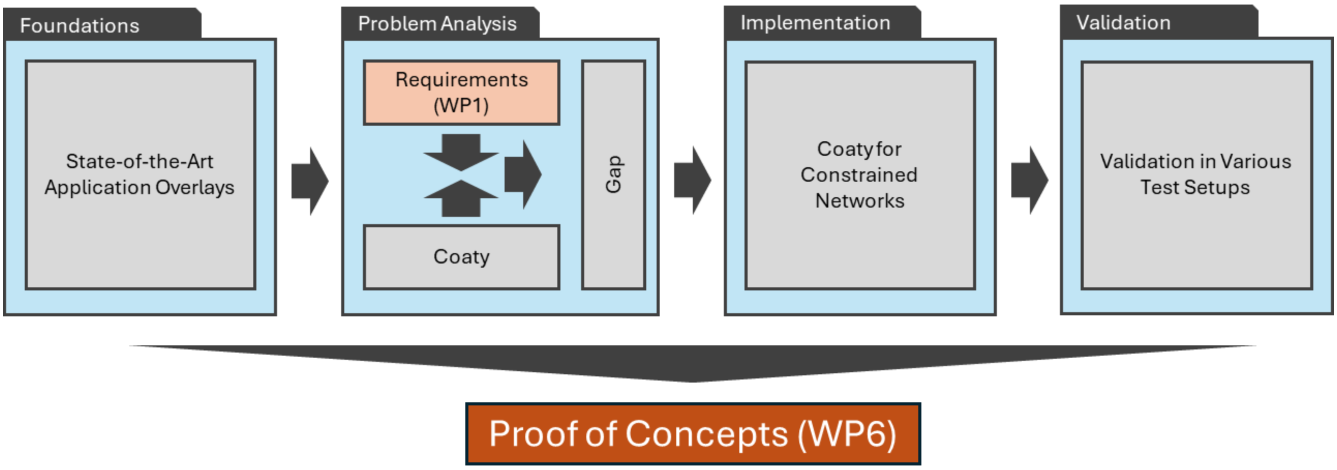

Figure 1 Methodology Overview

Application Overlay Networks

Recent advancements in communication, computation, and sensing, coupled with ongoing developments in artificial intelligence, have positioned data as central to all technological progress, often coined as the "new oil". To unlock the full potential of these technologies, seamless data exchange, synchronisation, and consistency across distributed collaborative nodes throughout the IoT-Edge-Cloud Continuum are essential. Application overlay networks are integral in realising this vision, facilitating robust data-centric operations.

This section offers a state-of-the-art analysis of the technological landscape surrounding application overlay networks. It delves into the core principles of these networks and outlines their role in meeting the OpenSwarm project requirements. The insights gathered here aim to lay the groundwork for a comprehensive understanding of the essential features of the Coaty framework that are pertinent to OpenSwarm.

Application Overlays Overview

To encapsulate the current state-of-the-art in application overlay networks, we propose a layered architecture for data-centric middleware[1] in collaborative distributed systems, as presented in Figure 2. This section elaborates on the functionalities provided by each layer, clarifying the roles they play in enhancing the functionality of application overlays. The subsequent discussion breaks down these layers and their respective functional decompositions, aiming to identify the alignment and gaps in existing solutions or technologies.

[1] In the remainder of this document the terms Application Overlay and Data-Centric Middleware will be used interchangeably.

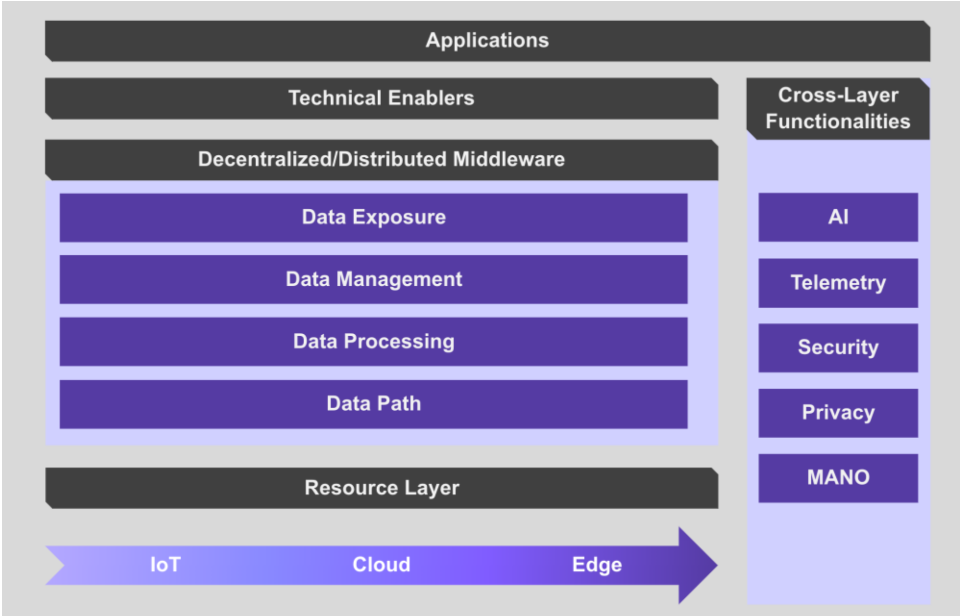

Figure 2: Components of an application overlay network

Architectural Overview

The functionalities of data-centric middleware can be grouped into four layers, as shown in Figure 2,. Each layer provides a different level of abstraction, building upon the previous layers - from interfacing with underlying infrastructure to exposing data-centric functionalities to application developers and system engineers. Moreover, certain functionalities that are orthogonal to middleware operations but crucial across all layers are grouped into a Cross-Layer Functionalities category.

Data Path Layer

This foundational layer abstracts low-level details, such as operating system and network stack, offering location and technology independence to the upper layers. As a result, developers can focus on the application logic without dealing directly with these underlying complexities. The data path layer facilitates data movement across the system's components, while ensuring adherence to agreed-upon QoS parameters. Key functionalities include:

- ***Infrastructure Abstraction:*** Implements a layer mapping data-centric primitives to underlying communication mechanisms (e.g., creating a new MQTT topic for one-to-one communication).

- ***Addressing:*** Ensures data delivery to correct destinations within the distributed system using data-centric primitives (i.e., identifying the data rather than the endpoints of the communication system), potentially integrated with security features (e.g., self-signed content).
- ***Forwarding:*** Middleware should forward data based on addressing information and configurable path selection strategies that consider factors, such as data context and QoS requirements (e.g., prioritisation, reservation, scheduling), and load balancing. Forwarding process can obey different communication patterns (e.g., one-to-one, one-to-many, many-to-many) and data delivery modes (at-most-once, at-least-once, exactly-once).

Data Processing Layer

Integrating communication and computation capabilities, this layer handles real-time, efficient, reliable, and context-aware data manipulation (e.g., aggregation, filtering, integration, calculations) and data persistence. It ensures data consistency in highly decentralised systems through state management and consensus algorithms. Key functionalities include:

- ***Data Manipulation:*** Performs computations, aggregations, filtering and transformations on incoming data to generate processed outputs. This function integrates communication and computation by placing data processing primitives along the data path.
- ***Data Contextualization:*** Enriches data with relevant metadata (e.g., timestamps, source identifiers, data format, location information), utilised in forwarding and processing stages.
- ***State management:*** Maintains system state information, fundamental for executing complex processing workflows where the behaviour of the system is defined by its current state.
- ***Caching:*** Stores temporary copies of data according to configurable caching policies, reducing latency and network traffic, coping with devices duty cycles and volatile resources, and realising complex workflows (e.g., wait until a given task is completed for an external input).

- ***Persistent Data Storage:*** Manages the persistence of data (raw and processed) and implement primitives for ensuring efficient read/write/query operations over distributed and heterogeneous storage technologies.
- ***Consistency:*** Ensures all components have a consistent and up-to-date view of the data, implementing the necessary mechanisms for conflict resolution and ensuring consistency across the overall distributed system.

Data Management Layer

Responsible for managing and orchestrating data-centric applications across distributed resources, this layer handles resource allocation and configuration based on application workflow requirements and data models. Key functionalities include:

- ***Data Modelling:*** Defines the data structure, format, encoding, relationships, and semantics crucial for configuring abstractions and ensure consistency and interoperability across distributed components.
- ***Data Flow Management \& Orchestration:*** Orchestrates the data flow between different components, scheduling data movement, transformation, and consistency. Allocates resources (e.g., memory, CPU, network bandwidth) to data-related tasks, optimising resource utilization based on data forwarding and processing requirements.
- ***Lifecycle Management:*** Oversees resource allocation, data forwarding strategies, data processing placement and execution order, and error handling to ensure the system operates reliably and scales effectively. It consumes monitoring information and adapts the system, being responsible for removing unnecessary forwarding and processing rules once an application is no longer being executed.
- ***Discovery:*** Collection of information about data sources and sync, as well as node resources (memory, CPU, storage, I/O, energy). Discovery mechanisms should support both reactive and proactive primitives, allowing the system to dynamically adapt to changes (e.g., node failures, additions, deletions, or mobility). Discovery information should include subscriptions, advertising, and resource allocation.

Data Exposure Layer

This layer makes the functionalities of the distributed data-centric middleware accessible through well-defined APIs, simplifying application development and deployment by abstracting the complexities of the underlying technologies. Key functionalities include:

- ***Application Programming Interfaces (API):*** Provides well-defined methods for interacting with the middleware, abstracting low-level communication details and exposing its capabilities and resources. Some of these interfaces should be intent-based to enable application developers to express high-level descriptions (i.e., intentions) of the expected system behaviour, which should then be translated into the corresponding data operations of the lower layers.
- ***Workflow definition:*** Enables the definition, processing, and decomposition of workflows for data processing and forwarding, which are then passed to the management layer for orchestration.
- ***Policy:*** Enables the enforcement of policies and rules for data sharing, authentication, authorization, and resource allocation and scheduling.
- ***Event handling and notification:*** Enables the notification of relevant events and keeps applications informed about system changes, allowing them to react to specific conditions.
- ***Targets and Constraints:*** Allows the definition of targets and constraints for data operations (e.g., desired data freshness, accuracy, timeliness, QoS).
- ***Federation:*** Supports operations across distributed domains or federated systems.

Cross-Layer Functionalities

This group incorporates orthogonal functionalities essential for the middleware's operation, covering aspects like AI-enhanced optimisation, telemetry for performance monitoring, and robust security and privacy measures to protect data and ensure compliance with regulations. More specifically:

- **Artificial Intelligence (AI):** plays a pivotal role in optimising data-centric middleware operations by solving complex challenges. For example, AI can analyse workload characteristics, network conditions, and resource availability to optimally locate processing tasks. Moreover, AI algorithms can learn from past actions (e.g., data processing placement, data forwarding strategy) to enhance future decisions, promoting system stability and reducing manual interventions. AI thus adds intelligence to middleware operations, increasing efficiency, reliability, and adaptability.

- **Telemetry:** Middleware requires comprehensive monitoring of data processes, performance, and system health. It tracks data flow, latency, errors, and resource usage. Combined with AI and management functionalities, telemetry supports a closed-loop middleware system that proactively manages operations and quickly identifies issues, enabling self-x capabilities (e.g., self-healing, self-scaling). Monitoring mechanisms should be tailored for highly distributed environments.
- **Security:** In data-centric middleware, security protects data and the system from unauthorised access, use, disclosure, disruption, modification, or destruction. This includes robust authentication and authorisation mechanisms to ensure data access is restricted to authorised entities. Additionally, securing communication channels and/or data prevents interception and tampering. AI can also be used to enhance security by enabling real-time thread detection and response.
- **Privacy:** Privacy mechanisms ensure data is handled according to strict privacy policies and regulations, such as General Data Protection Regulation (GDPR) in the European Union. This may involve anonymising data to remove personally identifiable information and enforcing strict data handling policies to ensure data is used only for its intended purpose.
- **Management and Orchestration:** Management involves administering middleware functionalities, ensuring they are correctly configured, deployed, and operating effectively across the IoT-edge-continuum. This includes diagnosing and resolving issues and updating or scaling functionalities as needed. Orchestration coordinates various middleware functionalities to operate seamlessly and efficiently, determining optimal placements across the continuum. It is important to note that this orchestration pertains to middleware components and not to distributed collaborative applications, which are managed by the Data Management layer.

Table 1, the main features of communication middleware as discussed above and identifies the value these features bring to OpenSwarm, building upon the requirements identified in WP1.

Table 1 Application Overlay Value Proposition

Feature	Value
High-Level Abstraction	The data-centric middleware serves as an ideal abstraction of the underlying infrastructure, masking device heterogeneity and ad-hoc topologies. This simplification eases system use and management, promotes efficient resources utilisation, and supports higher-level communication requirements, making the system more flexible, maintainable, and scalable.
Tailored Communication Patterns	Communication patterns can be customised to meet specific system needs and capabilities. This adaptability allows the system to respond dynamically to changing conditions, thus optimising performance to meet diverse requirements and objectives.
Location Independent Primitives	By focusing core communication functionalities on the data itself, the system operates independently of the specific locations of resources, enhancing robustness, availability, and flexibility. This feature ensures continuity of operations despite changes or failures in the system.
Flexible Operation	By fully leveraging the IoT-Cloud-Continuum, optimisation mechanisms can be applied to utilise the most suitable resources at any given time. Adapting optimisation criteria (e.g., quality of service, energy efficiency) allows the system to meet various

	requirements or objectives, making it versatile across various use cases and environments.
Seamless Discovery	Data-centric communication primitives also facilitate discovery, allowing the network to identify and incorporate new mobile objects (e.g. robots), expanding the coverage and capabilities of the continuum. This simplifies bootstrapping and matches resource utilisation with data abstractions and node configurations.
Decentralized Storage	With data central to operations and the ability to handle heterogeneous storage across the continuum, the middleware can seamlessly offer caching and long-term storage functionalities. This feature enhances the performance and efficiency of the system, also ensuring high robustness and availability.
Context-aware Operation	Context-based message routing can be integrated to dynamically manage context, enhancing system adaptability.
Content-based Security	Placing data at the core of the middleware functionalities allows shifting security focus to the content itself, moving away from traditional channel-based security. This shift increases the system's flexibility to utilise the full continuum of resources.

Technological Landscape

NATS

NATS[2] is a data-centric middleware designed for modern distributed systems. It operates through interconnected NATS server processes, which can scale from a single

[2] https://nats.io/

process to a global super-cluster, including leaf nodes that can function even when temporarily disconnected.

Messages in NATS are addressed using subject strings, which can be organized hierarchically with the '.' character and wildcards for flexible subscriptions. NATS supports different data delivery modes: at-most-once (Core NATS), at-least-once, and exactly-once (via JetStream). JetStream provides message persistence, replay, and encryption at rest, using an optimized RAFT algorithm for clustering.

NATS supports various communication protocols (e.g., MQTT, NATS, NATS with TLS) and patterns like Publish-Subscribe and Request-Reply. Dynamic queue groups allow scalable responder groups. The NATS Execution Engine (Nex) deploys and runs workloads, including long-running services and short-lived functions (JavaScript or WebAssembly [1]).

Observability in NATS is straightforward, allowing monitoring of requests and responses. JetStream publishes health and state advisories, which can be stored in JetStream Streams if desired.

Kafka

Kafka[3] [1] is an open-source distributed event streaming platform. It consists of ***brokers***, that form the storage layer; ***Kafka Connect servers***, that enable the integration with other systems (e.g., databases, other Kafka clusters); and ***clients***, that read, write, and process streams. Kafka follows a ***push-pull communication model***, where data is pushed to the broker from the producer and pulled from the broker by the consumer.

Client applications are either producers, that publish (i.e., write) events to Kafka, or consumers that subscribe to (read and process) these events. In Kafka, ***producers and consumers are fully decoupled*** and agnostic of each other. Kafka provides various data delivery modes (i.e., ***at-most-once, at-least-once, exactly-once***).

Events are organized and durably stored in ***topics*** that are multi-producer and multi-subscriber (i.e., supports ***M:N communication models***). Events in a topic can be read as

[3] https://kafka.apache.org/

often as needed and are not deleted after consumption. Instead, it is possible to define for how long Kafka should retain events for a particular topic. Topics are partitioned over ***buckets*** located on different Kafka brokers to the increase the system ***scalability*** by allowing client applications to both read and write the data from/to many brokers at the same time. Also, topics can use replication to ensure fault-tolerance and highly-availability.

In Kafka, data is mostly consumed in a streaming fashion using tail reads (i.e., levaring OS's page cache to serve the data instead of reading from disk). Moreover, Kafka implements a tiered storage approach, where a Kafka cluster is configured with two tiers of storage (i.e., local and remote). The local tier uses the local disks on the Kafka brokers to store the log segments and the remote tier uses external storage systems (e.g., HDFS or S3), to store the completed log segments.

Kafka relied on Zookeeper for cluster coordination ensuring robust synchronization within the distributed system. However, Zookeeper is being replaced by Kafka Raft (kRaft) consensus protocol.

Flink

Apache Flink[4] [2] is a distributed processing engine for stateful computations over unbounded and bounded data streams. The Flink runtime consists of two types of processes: a ***JobManager*** and one or more ***TaskManagers***.

The JobManager decides when to schedule the next task (or set of tasks), reacts to finished tasks or execution failures, coordinates checkpoints, and coordinates recovery on failures, among others. This process consists of three different components: (i) ***ResourceManager***: responsible for resource de-/allocation and provisioning; (ii) ***Dispatcher***: provides a REST interface to submit Flink applications for execution and starts a new JobMaster for each submitted job; (iii) ***JobMaster*** is responsible for managing the execution of a single JobGraph.

[4] https://flink.apache.org/

The TaskManagers (also called workers) execute the tasks of a dataflow, and buffer and exchange the data streams. The smallest unit of resource scheduling in a TaskManager is a task slot. The number of task slots in a TaskManager indicates the number of concurrent processing tasks. Each task slot represents a fixed subset of resources of the TaskManager. Slotting the resources means that a subtask will not compete with subtasks from other jobs for managed memory, but instead has a certain amount of reserved managed memory. Currently slots only separate the managed memory of tasks (i.e., there is no CPU isolation)

Coaty

Coaty[5] [3], which is the key solution targeted in this deliverable, is a data-centric middleware for distributed collaborative applications. Coaty uses event-based communication flows with **one-way/two-way** and **one-to-many/many-to-many** event patterns to realize decentralized prosumer scenarios.

Coaty provides a set of event-based communication patterns to discover, query, share, and update data on demand in a distributed system, and to request execution of context-filtered remote operations. Coaty combines the characteristics of both classic **request-response** and **publish-subscribe** communication. Coaty provides Smart Routing of IoT (sensor) data, or also called **IO Routing**, where data is dynamically routed from sources to actors, i.e., consumers based on context. One of the unique features of Coaty communication is the fact that a single request in principle can yield multiple responses over time, even from the same responder.

[5] https://coaty.io/

Interplanetary Filesystem (IPFS)

IPFS[6] [4] is a peer-to-peer distributed file system that takes from successful ideas from previous peer-to-peer systems (e.g., DHTs, BitTorrent, Git). It uses content addressing to uniquely identify files and enables decentralized file storage, retrieval, and sharing across a network of peers. IPFS is often used for building decentralized applications and sharing and versioning large datasets.

IPFS is peer-to-peer; no nodes are privileged. IPFS nodes store IPFS objects in local storage and connect to each other and transfer objects. These objects represent files and other data structures. Nodes are identified with a NodeId resulting of the cryptographic hash of a public key (not bounded to a particular hash function, instead there is a short header specifying the hash function used. IPFS nodes rely on a DHT routing system to locate specific peers and objects.

To ensure that IPFS can be used in overlay networks it is not restricted to a particular network protocol, instead it runs on top of various underlying network protocols, ensuring internet-wide connectivity. IPFS can adapt to the underlying network stack (e.g., to implement a missing functionality on a given stack).

IPFS nodes rely on a DHT routing system to locate specific peers and objects. Data distribution happens by exchanging blocks with peers using BitSwap, a novel block exchange protocol, BitTorrent inspired. On top of this, IPFS builds a Merkle DAG of content-addressed immutable objects with links and defines a set of objects for modeling a versioned filesystem on top of the Merkle DAG. IPFS uses a naming scheme from Self-Certified Filesystems (SFS) to construct self-certified names that are mutable.

[6] https://ipfs.tech/

Internet Computer

The Internet Computer (IC)[7] [5] is a decentralized blockchain-based platform for the execution of general-purpose decentralized applications (dapps) in an efficient and secure manner. The IC offers decentralized stateful serverless computation over untrusted, independent datacenters. Application developers deploy dapps (equivalent to serverless function workflows) on the IC without the cumbersome process of resource management, just like in serverless environments. The dapps interact with each other and with end-users. Each dapp is composed of canisters, the smallest units containing code and data, an immediate equivalent to serverless functions. Developers deploy stateful canisters that serve calls either to end-users or other canisters.

The IC programming model is like serverless clouds, with applications written in modern languages such as Rust, Python and Motoko. The IC canister code is compiled down to WebAssembly, which is executed under a sandboxed virtual machine on the IC nodes. Any language that can be compiled down to WebAssembly could also be used. Coupled with canister statefulness, the IC programming model is inspired by an event-driven, actor-based model, where application programmers implement functionality that responds to messages from users or other canisters.

The protocol stack run by the nodes of the IC consists of 4 layers: (i) the ***peer-to-peer layer*** enables information exchange among nodes; (ii) the ***consensus layer*** validates and orders messages to ensure information consistency across all replicas; (iii) the ***message routing layer*** selects the destination canister and enqueues the message for processing; (iv) the ***execution layer*** takes messages from canister input queues and executes the corresponding Wasm function with the message as payload.

Oakestra

Oakestra [6] is a hierarchical, lightweight, flexible, and scalable orchestration framework for edge computing. It proposes a novel federated three-tier hierarchy to consolidate

[7] https://internetcomputer.org/

edge infrastructures: (i) the ***root orchestrator*** provides a centralized control plane for managing cluster resources; the ***cluster orchestrators*** are logically similar to the root but responsible of resource management within the local cluster; the ***worker nodes*** are edge computing servers within clusters that execute the services.

Oakestra offers a lightweight and modular implementation compatible with most popular cloud technologies and is extensible by design. It implements delegated task scheduling where the root makes only coarse-grained cluster selection, and the fine-grained resource placement is done by the clusters. Additionally, it offers a novel semantic overlay networking that leverages dynamic routing policies transparently enforced via semantic service addressing to enable seamless edge-oriented load balancing policies.

As a result, Oakestra can flexibly consolidate multiple infrastructure providers and support applications over dynamic variations at the edge and application providers can be abstracted of the underlying complexity of the edge and deploy services by specifying high-level constraints (e.g., hardware, latency, localization).

DAPR

DAPR[8] is an open-source, portable, serverless, event-driven runtime that simplifies building resilient, stateless, and stateful distributed applications that run on the cloud and edge. It provides abstractions for common distributed system tasks and supports various languages and developer frameworks.

Dapr offers distributed system building blocks through standard HTTP verbs or gRPC interfaces. These interfaces are exposed using a sidecar architecture, either as a container or as a process, providing separation of the application logic for improved supportability.

Dapr provides Publish-Subscribe communication with at-least-once message delivery guarantee, message TTL, consumer groups and other advance features. Dapr support

[8] https://dapr.io/

workflows to define long running, persistent processes or data flows that span multiple microservices.

Dapr provides many capabilities in its actor runtime, including concurrency, state, and life-cycle management. Threading, state consistency models and message delivery patters can be tailored to the application requirements.

Dapr provides a Web-UI dashboard for information visualization, monitoring tools for visibility into the Dapr system services and sidecars, and observability capabilities to provide insights into your application, such as tracing and metrics.

Actyx

Actyx[9] is a decentralized event database, streaming and processing engine that implements local-first cooperation. It enables a novel way of building software that runs across multiple networked devices, leveraging local capabilities and removing dependency on central components (e.g., servers, databases, or cloud).

Processes are programmed as autonomous small pieces of code called ***local twins***. These local twins are digital twins that resides, perceives, and acts locally. They publish and consume events, and develop states based on these events. Actyx automatically synchronizes the twins within the local network.

After you have programmed the local twins, you create local computing environments using edge devices. The local computing environment provides the infrastructure necessary for running local twins. Due to the local interaction of the twins, there is no dependency between environments.

In addition, Actyx offers a framework running on top Actyx that enables easy choreography of distributed and heterogeneous a swarm to execute a sequential workflow without complex and expensive coordination of states. An Actyx swarm is a mesh of nodes communicating in a local network. A swarm may have 1 to N nodes in it.

---

[9] https://developer.actyx.com/

A swarm is defined by a single property, the so-called swarm key. In order to participate in a swarm, a node must have the secret swarm key.

Akka

Akka[10] is a toolkit and runtime for building highly concurrent, distributed, and fault-tolerant systems. It's based on the Actor model and provides abstractions to ease writing correct concurrent, parallel and distributed systems. These abstractions enable seamless multi-threaded behavior (e.g., without the use of low-level concurrency constructs like atomics or locks); transparent remote communication between systems and their components; and clustered, high-availability, elastic architecture, that scales on demand. Overall, it solves difficult distributed/parallel systems problems in a uniform programming model where everything fits together tightly and efficiently.

Airflow

Apache Airflow[11] is an open-source platform for orchestrating complex workflows. It allows users to define, schedule, and monitor data pipelines, making it popular for ETL (Extract, Transform, Load) tasks and data processing.

Airflow implements the Workflow as a Code paradigm, where workflows are defined using Python code and Workflow parameterization is built-in leveraging the Jinja templating engine. This further enables workflows to leverage well-known software development processes and pipelines (e.g., version control, collaboration, testing).

Airflow supports different deployment strategies (from standalone to distributed deployments), contains operators to connect with numerous technologies and components are extensible to easily adjust to your environment.

---

[10] https://akka.io/

[11] https://airflow.apache.org/

Data Distribution System (DDS)

DDS [8] is an open standard for real-time, scalable, and secure data communication in distributed systems. It is commonly used in the Industrial Internet of Things (IIoT) and other mission-critical applications. Unlike traditional message brokers, DDS follows a peer-to-peer communication model, allowing direct data exchange between publishers and subscribers without intermediaries1. It's optimized for distributed processing and is well-suited for scenarios where low-latency and high-throughput communication are essential.

Named Data Networking (NDN)

Named Data Networking (NDN)[12] [9] is one of the more prominent Future Internet Architecture proposals. Resulting of an evolution of the Content-Centric Networking (CCN), it aims to replace the current IP-based model. The basic operational principle of NDN is to address data itself rather than the location where the Data is available. NDN follows a request-response communication pattern and uses hierarchical names to identify the content being exchanged. This approach allows for efficient content retrieval, in-network information caching, and content-based security.

Zenoh

Zenoh[13] [10] is a novel data-centric middleware designed for edge computing and IoT environments. It combines features from publish-subscribe systems, databases, and content-centric networking. Zenoh supports both brokered communication (similar to MQTT and Kafka) and peer-to-peer communication (like DDS). Zenoh-Pico extends Zenoh to microcontrollers and embedded devices. Zenoh provides location-

[12] https://named-data.net/

[13] https://zenoh.io/

transparent abstractions for high performance pub/sub and distributed queries across heterogeneous systems. Additionally it provides universal abstractions for cloud-to-device data-flow programming.

Design and Implementation

Coaty

Using Coaty [koʊti] as middleware (an application overlay network), you can build distributed applications from decentralised application components, called Coaty agents, which are loosely coupled and communicate with each other. The framework primarily focuses on IoT prosumer scenarios where smart agents act autonomously, collaboratively, and in an ad-hoc fashion.

Coaty Framework

The Coaty framework for collaborative open systems is based on a lightweight and modular architecture. This framework relies on the following key components:

- A local network (e.g., WLAN, LPWAN) that connects the participating systems in prosumer scenarios.
- Node and service **discovery**, like multicast DNS (mDNS).
- A **Publish/Subscribe** (Pub/Sub) service (e.g., MQTT Broker).
- An **object model** that supports generic and universal communication, allowing each collaborative system to model domain-specific characteristics, as well as tasks as objects.
- Event-based **communication patterns** to discover, query, share, and update data on demand within a distributed system.

Coaty communication patterns

Coaty achieves flexibility in distributed systems by decoupling communication endpoints while maintaining all types of communication flows.

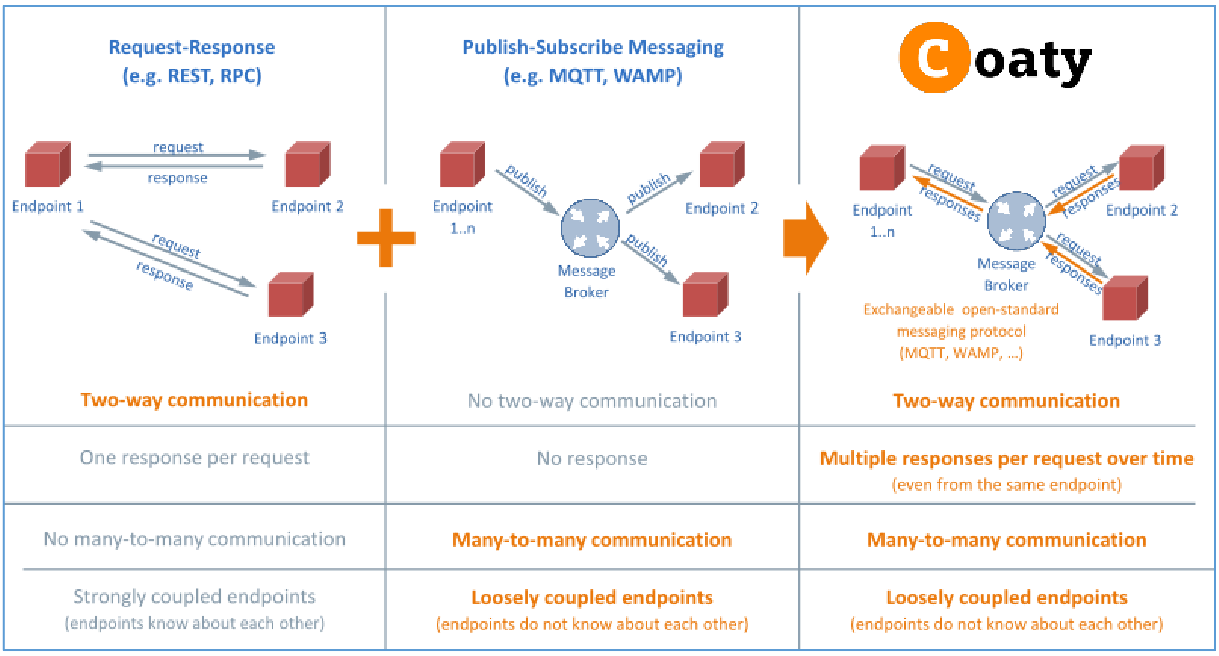

Figure 3: Coaty Communication foundation

The framework uses event-based communication flows with one-way/two-way and one-to-many/many-to-many event patterns to realise decentralised prosumer scenarios. It combines the characteristics of both classic request-response and publish-subscribe communication models. Unlike traditional client-server systems, all participants in the system are equal, meaning they can act as both producers/requesters and consumers/responders.

Coaty communication event patterns

Coaty provides a set of event-based communication patterns to discover, query, share, and update data on demand in a distributed system, as well as to request the execution of context-filtered remote operations.

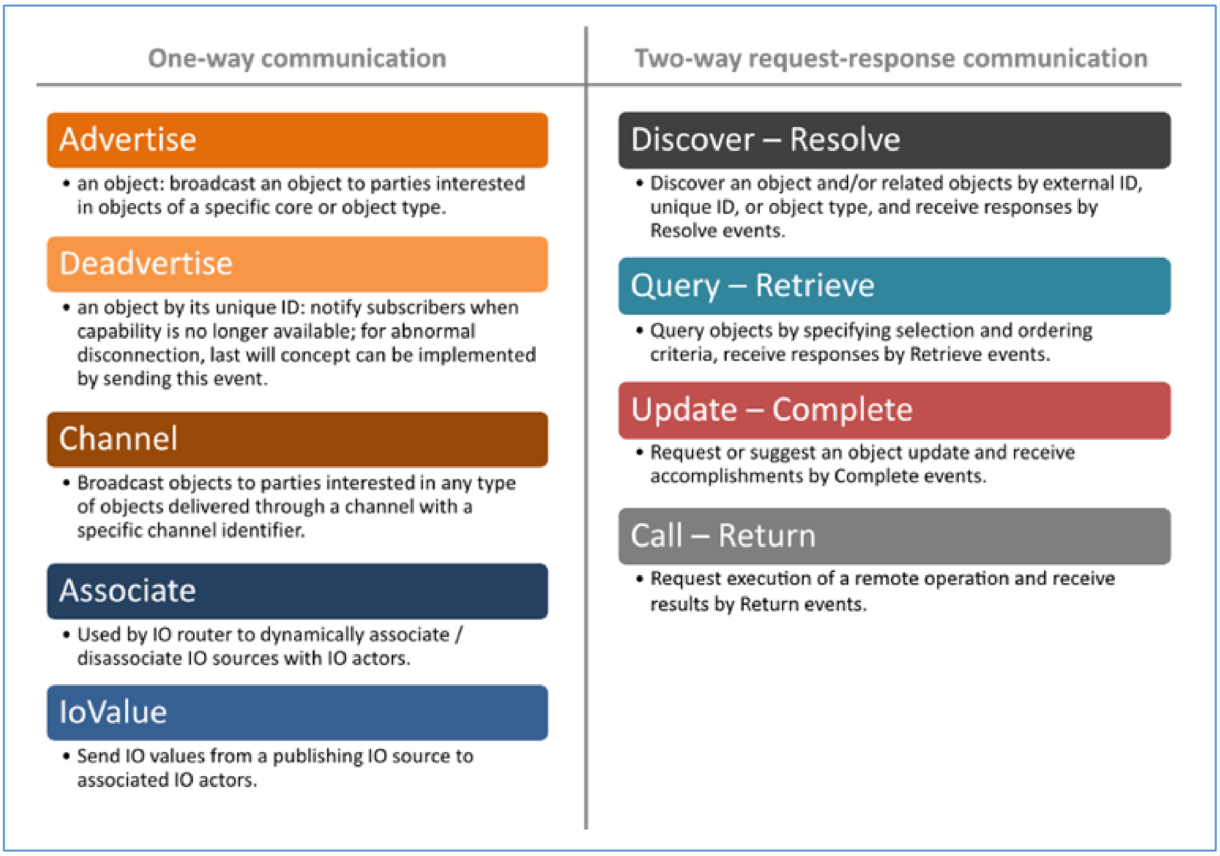

Figure 4: Coaty Communication event patterns

Coaty's communication event patterns are built on top of interchangeable open-standard publish-subscribe (pub/sub) messaging protocols, which can either be broker-based (e.g., MQTT or WAMP) or brokerless (e.g., DDS or peer-to-peer approaches). Through its communication bindings, one can choose a specific messaging transport for the Coaty application while keeping the event patterns and application code unaffected. By selecting a WebSocket-aware communication binding, Coaty agents can also natively run in mobile and web browsers.

With the help of Reactive Programming, all Coaty event patterns can be programmed in a simple, uniform way. The Coaty framework is available as open-source software. The Type-/JavaScript implementation can be found at: https://github.com/coatyio/coaty-js. For additional information, you may visit: https://coaty.io/.

Evolving Coaty for constrained environment

This section derives the main OpenSwarm requirements place upon the Coaty framework and provides detail of its evolution into a new release of Coaty for constrained environments.

Assessing the Coaty capabilities against the OpenSwarm requirements

The capabilities of Coaty, as discussed in the previous section, are assessed against the OpenSwarm requirements from D1.2.

First, the requirements are classified based on their relevance to an Application Overlay Network (column: not relevant). This classification is grounded in the general perspective on Application Overlay Networks, as presented in the previous section. Excluded from this analysis are requirements (column: required) that need to be fulfilled by a lower-level component of the system (such as firmware) or directly by the application itself.

Next, we assess how Coaty meets the relevant requirements (columns: fully, partially, or not supported). This assessment leads to a shared vision of what could be considered as the “constrained Coaty” version.

Table 2. Requirement analysis.

	Title	page	not relevant	required	fully supported	partially supported	not yet supported	needs to be clarified
	User Experience		X					
	Test Coverage				X			
Error! Reference	Security				X			
	Secure Joining				X			

Table 3. Consolidated requirements assessment.

Requirement ID	Title	page	not relevant	required	fully supported	partially supported	not yet supported
	Communication Requirements						
Req.1	Regulatory Compliance	36	X				
Req.2	General Communication Needs	37		X			
Req.3	Sensor Network Integration	37		X			
Req.4	Communication patterns	38			X		
Req.5	Network Self-Optimization	38		X			
Req.6	Interference Mitigation	38	X				
Req.7	Ad-Hoc Communication	39		X			
Req.8	Addressing	39			X		
Req.9	Quality of Service (QoS)	40		X			
	Components						
Req.10	Swarm Gateway / Swarm Connector	40	X				
Req.11	Sensor Requirements	40	X				
Req.12	Mobile Devices	41	X				
	Energy Consumption						
Req.13	Power / Energy Consumption	41					X
Req.14	Battery Lifetime	42	X				
Req.15	Energy Harvesting and Efficiency	43	X				
	Configuration						
Req.16	Discovery	43			X		
Req.17	Auto Configuration / Negotiation	44		X			
Req.18	Adaptability / Change detection	44			X		
Req.19	Self-description / Awareness of own needs	45		X			
Req.20	Localization	46		X			
	System Management						
Req.21	Application Deployment	46				X	
Req.22	System and Device Management /	47		X			
	Application Performance Monitoring					X	
	Data Processing						

Requirement ID	Title	page	not relevant	required	fully supported	partially supported	not yet supported
Req.23	GDPR Compliance	47				X	
Req.24	Information Caching	47				X	
Req.25	External Information Storage	48				X	
Req.26	Distributed Data Fusion	48	X			X	
Req.27	Context Identification	48	X			X	
Req.28	State Synchronization	48				X	
Req.29	Latency / Real-time Data Processing	48				X	
	Use Case Functions						
Req.30	Boat Detection	49	X				
Req.31	Reporting of Boat Detection	49	X				
Req.32	Dynamic Object Tracking	50	X				
Req.33	Proximity Detection	50	X				
	System Requirements						
Req.34	Interoperability	50				X	
Req.35	Scalability	51			X		
Req.36	Maintainability	51				X	
Req.37	Security	51				X	
Req.38	Flexibility	52				X	
Req.39	Robustness	52				X	
Req.40	Reliability / Availability	53				X	
Req.41	Ease of use / Usability / User Interface	54			X		
Req.42	Cost-Effectiveness	54	X				
Req.43	Test Coverage	54			X		
Req.44	Documentation and Support	55			X		

Of the 45 (Req.22 divided in Req.22a and Req.22b) consolidated OpenSwarm requirements are 24 requirements relevant for the Application Overlay Network.

Of these, 8 are fully and 15 are partially met by the current Coaty implementation.

One requirement is not and 15 are only partially supported by the current Coaty implementation which leads to the implementation of a first constraint Coaty version.

First implementation release of DDA as Coaty successor for constraint devices

To address the OpenSwarm requirements that are either unmet or only partially fulfilled by Coaty, in particular Req. 13 (Power/ Energy Consumption), Req. 22 (Application Performance Monitoring), Req. 34 (Interoperability), Req. 36 (Maintainability), Req. 38, (Flexibility), and Req. 39 (Robustness), we developed a first implementation release of Coaty for constraint devices (with reduced footprint in go, gRPC interface, and MQTT as transport mechanism). This Coaty variant was named Distributed Data Agent (DDA). It is based on MQTT as a transport mechanism and includes the following features:

- **Reduced footprint:**
    - Implementation in Go
    - gRPC interface support
    - minimalised set of communication patterns
    - exclusion of the object model
- **Basic telemetry data collection** integrated with OpenTelemetry
- **Deployment options** as a Go library or using the sidecar pattern

The DDA framework is available as open-source software. The implementation can be found at: https://github.com/coatyio/dda

Performance Analysis

Different setups were considered for the evaluation of the DDA, ranging from a 6TiSCH constrained network to a heterogeneous 6TiSCH–WiFi network, which included sensor nodes, IPCs and consumer devices.

The test setups are structured into the following categories:

- Heterogeneous overall setup
- Single / two Smart Mesh IP networks (separated as well as coupled via DDA)
- Smart Mesh IP Gateway / Tunnel
- IP Routing

Test Setups

The test setups are organized as follows:

1. **Objective**: Each test begins by a clearly defined objective that states the purpose or goal, explaining why the particular setup is being conducted. This helps to provide clarity on what the test aims to achieve.
2. **Setup**:
   - **Device Representation**: The different devices involved in the test are described, representing a variety of network types (heterogeneous networks).
   - **Software Stacks**: The software running on each device is detailed, as the software stacks may differ depending on the device type.
   - **Connections**: The connections between the devices are explained, with a focus on how different networks and device classes are connected.
   - **Test Applications**: For each device, the specific test application used is outlined, along with the tasks the application handles and its interaction pattern based on the Coaty successor.
3. **Results**: The findings from each test are presented, highlighting key outcomes and any significant data collected.
4. **Discussion**: The results are analysed and discussed, interpreting their meaning in relation to the test objectives and considering broader implications or insights gained from the setup.

Finally, the overall result is summarised. In the discussion section, the next steps and further procedures are outlined based on the findings.

The general goal is to develop an overall heterogenous scenario that covers all network and device types and classes.

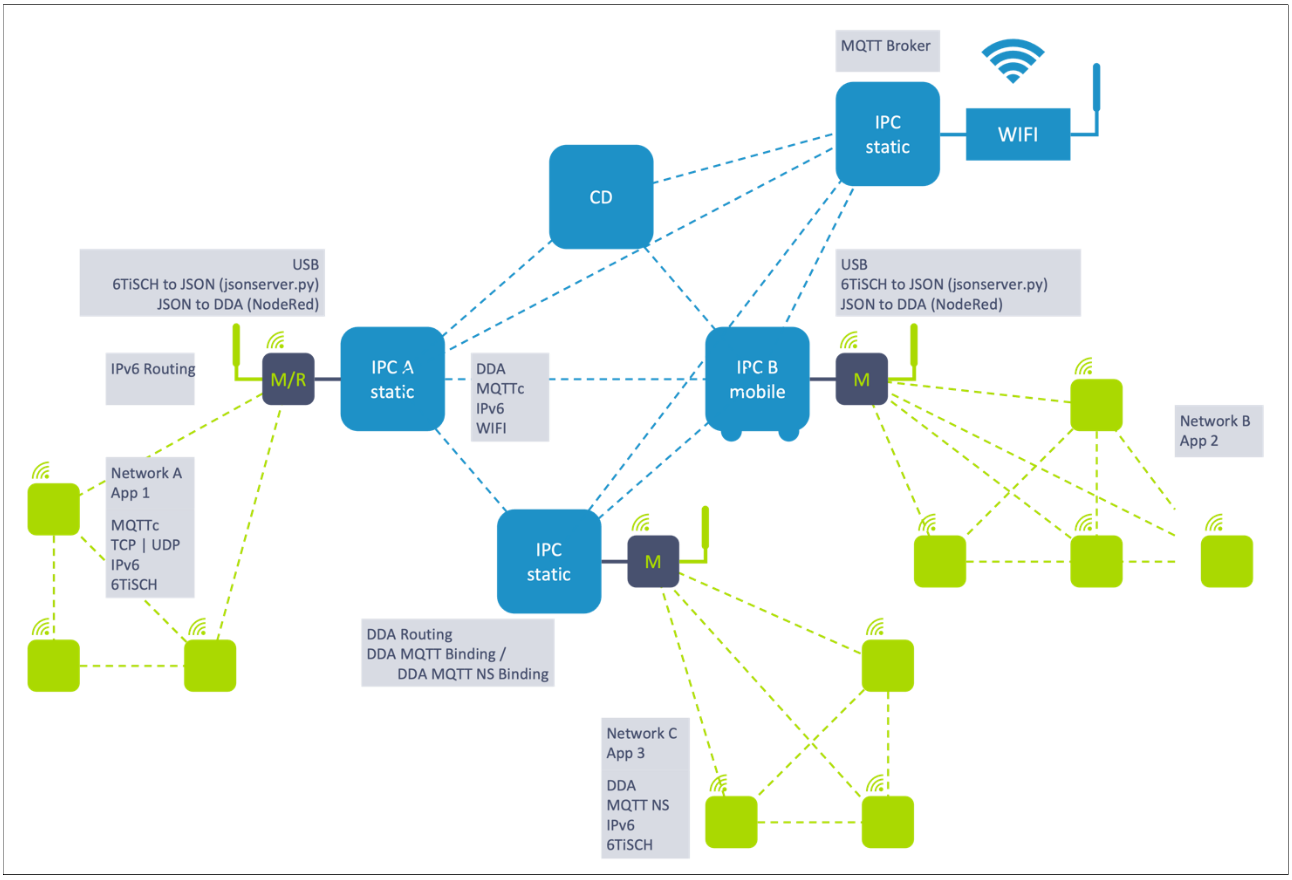

Figure 5: Total test setup 1 – Overall heterogeneous scenario

Test setup 1 – Single SmartMesh IP / 6TiSCH (AIOT) cluster

Objective

Gain experience about the behaviour of SmartMesh IP network by clarifying the questions:

- How long does it take until a connection to a manger is established?
- How long does it take,
    a) If the node is connected for the first time?
    b) if the node was already connected to the manager?

Setup

The setup consists of three SmartMesh IP nodes (N1, 2, 3), an SmartMesh IP network manager (M), connected via USB with an IPC.

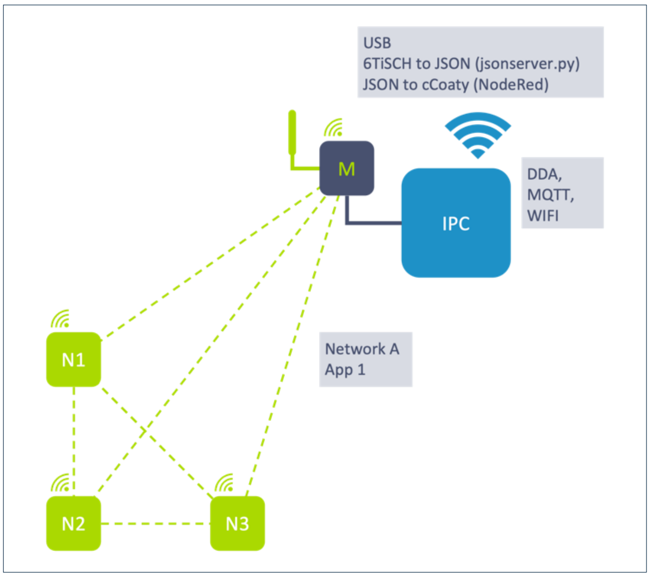

Figure 6: Smart Mesh IP (AIOT) setup

Result

The time it takes for a node to be known in a network varies and depends on the power off period. A setup is required with which more complex measurements can be carried out automatically and repeatably.

Such a measurement setup is designed, constructed and used for the test-setup 2. The test that is carried out with the test setup 2 includes the measurements for this test setup 1.

Discussion

The time it takes for a node to be known in a network can be too long to implement individual scenarios.

Test setup 2 – Two SmartMesh IP / 6TiSCH (AIOT) cluster

Objective

Gain experience about the behaviour of SmartMesh IP networks by clarifying the questions:

- What is the delay for network join in different networks?
- How long does it take until a connection to manger B is established, when the node already was connected to the manager A?
- Is there a different in duration between the first and the following connection setups?
- What temporal behaviour is evident when running through the following setup, configurations and schemes?

Setup

The test setup is based on setup 1 and was extended by a second SmartMesh / 6 TiSCH network.

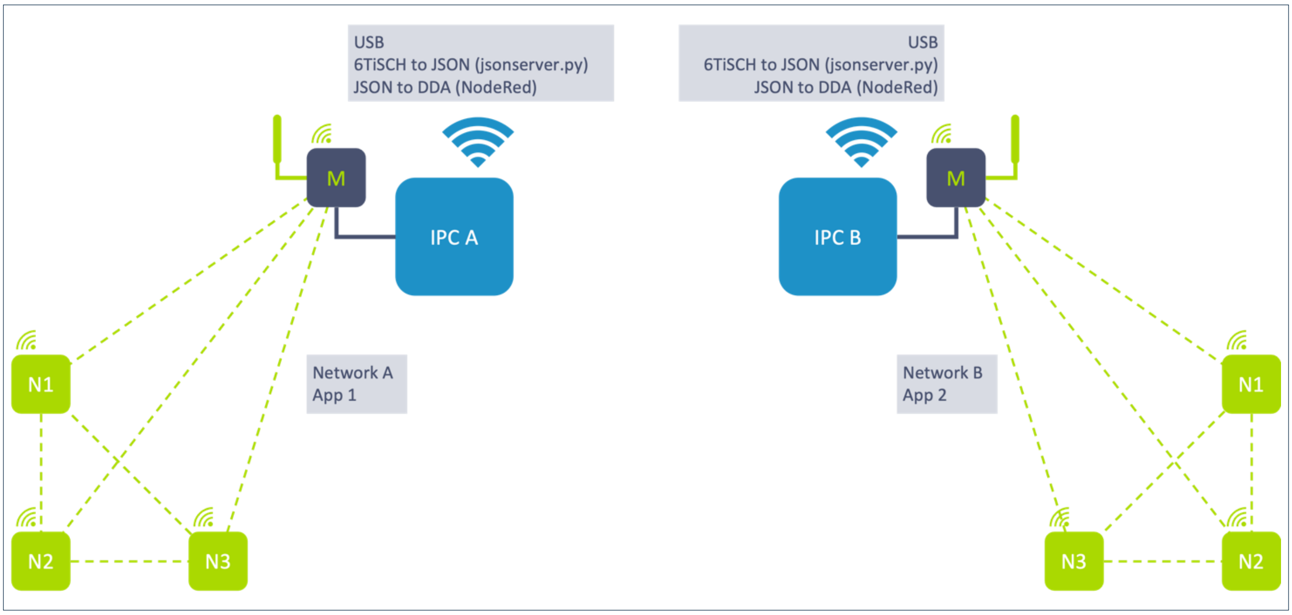

Figure 7: Extended Smart Mesh IP (AIOT) setup

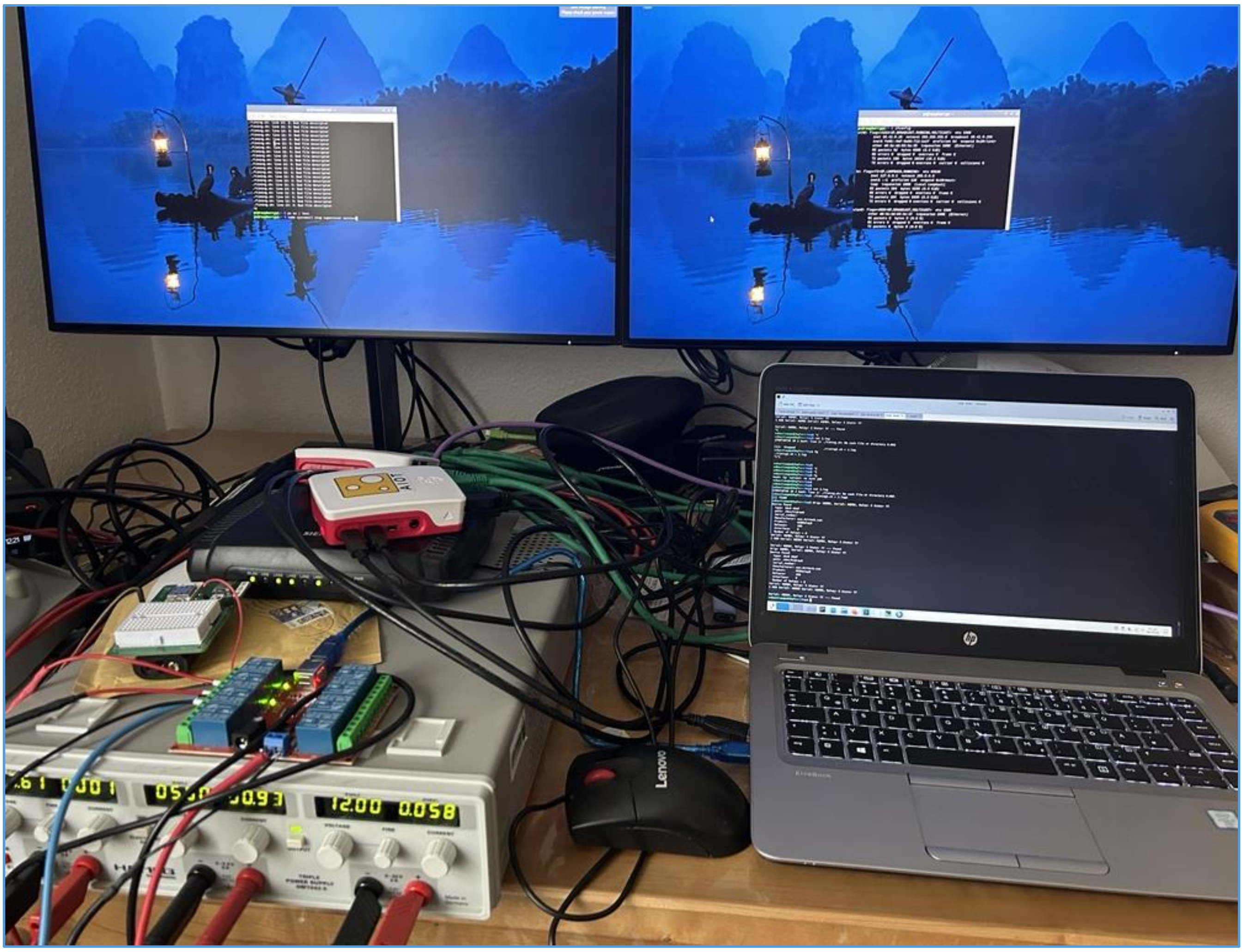

Figure 8: Physical structure of the test environment

In order to answer all questions, the setup shown in figure 8 was set up. This test setup allows to configure and switch the devices. The test was performed four dimensional. Three dimensions are binary, while one dimension (the power off time) was evaluated for different values.

The combination of the different test dimensions was bit-encoded according to the following scheme.

- 1 second master
- 2 bbmode
- 4 JDC255 mote (join duty cycle 255)
    - Second master means that mode is offered a different master for each connection attempt.
    - bbmode means that backbone mode is activated on master.

    - JDC255 mote means, the mode used is configured to have a Join Duty Cycle of 255 set.

The test was implemented as shown below as pseudo-code

- Loop forever
    - Loop over all modes
        - Loop over all delay times
            - Execute a hundred times
                - Turn on master (depending on mode)
                - Sleep delay time
                - Turn on mote (depending on mode)
    - When connected record time and turn off

Results

The main results are summarized in the following table. The table shows the join delay median (over 100 runs) in seconds for different setups and power-off times. The first column represents the configuration mode; the second column provides the binary representation of the main three configuration parameters (i.e., DC – Duty Cycle; BB – Backbone-mode; M – Switching between Masters enabled); the last column presents the results for the different power-off time configurations. For each combination of configuration parameters, the join delay is represented.

For example the mode 0, means, that Join Duty Cycle was configured to be default (in contrast to always on/JDC255), Backbone-mode was not enabled and there was no switch between masters, which resulted in a median join time of 8 seconds, when the mode was powered off for 50 seconds and 38 seconds, when the mote was powered off for 100 seconds.

Table 2: Median Join Delay in seconds for different setups and power-off times

				Join delay [s]				
	Setup			Power-off time [s]				
#	DC	BB	M	50	100	200	500	1000
0	0	0	0	**8**[1]	**38**	**35**	**42**	**36**
1	0	0	1	**42**	**37**	**37**	**37**	**34**
2	0	1	0	**8**[1]	**37**	**46**	**41**	**35**
3	0	1	1	**32**	**34**	**35**	**37**	**41**
4	1	0	0	**1**[1,2]	**1**[2]	**1**[2]	**1**[2]	**1**[2]
5	1	0	1	**64**[3]	**10**[3]	**1**[2]	**1**[2]	**1**[2]

Based on that three major observations could be identified:

1) Reconnect to the same master is fast if interruption was short enough.
2) A higher Join Duty Cycle causes instant connection in most cases,
3) when, connecting to a new master after some timeout

For a scenario with mobile motes and changing networks, which limited timeframes to connect to a smart mesh IP network, it is necessary to set a timeout for connecting to the same master to a low value while setting the JDC to a high value.

In principle and in addition to that:

- A node can only be connected to one master at a time.
- This limits the applicability of DDA in multi-master deployments.

Discussion

The results clearly show that the network join delays for the same configuration depend on the PowerOffTime in a way, that there is a threshold, below which a connection to the same master as used before is fastest, while above connecting to any master is faster. Obviously, motes try to reconnect for about two minutes to the known master, before joining a different master's network.

As it could be expected the join time highly depends on the Join Duty Cycle, which expresses the share of time, where the motes tries to find a network spending 5mA current for the radio. Backbonemode has no effect at all. During design of an application therefore most interesting questions would be the best value of the Join Duty Cycle, which is a compromise between energy consumption and delay, which importance is based on typical number of reconnections required and duration of impossible reconnect.

It should be clearly stated that Time used for joining a network is not network formation time.

For certain use cases it might be interesting to interconnect independent islands of local (ethernet) connected devices using SmarhMesh IP. A setup like shown in Figure 9 could serve the purpose. Therefore, additional tests were performed to understand whether it was possible to connect multiple IPCs at the same moment in time, as shown in Figure 9. The result of this tests indicated that it will not be possible to use a SmartMesh IP network as medium for such a deployment scenario. The reason for this is that RPL (Routing protocol for low power and lossy networks) only supports one router, as otherwise it would become necessary to maintain different routes for different target addresses, making the protocol less lightweight.

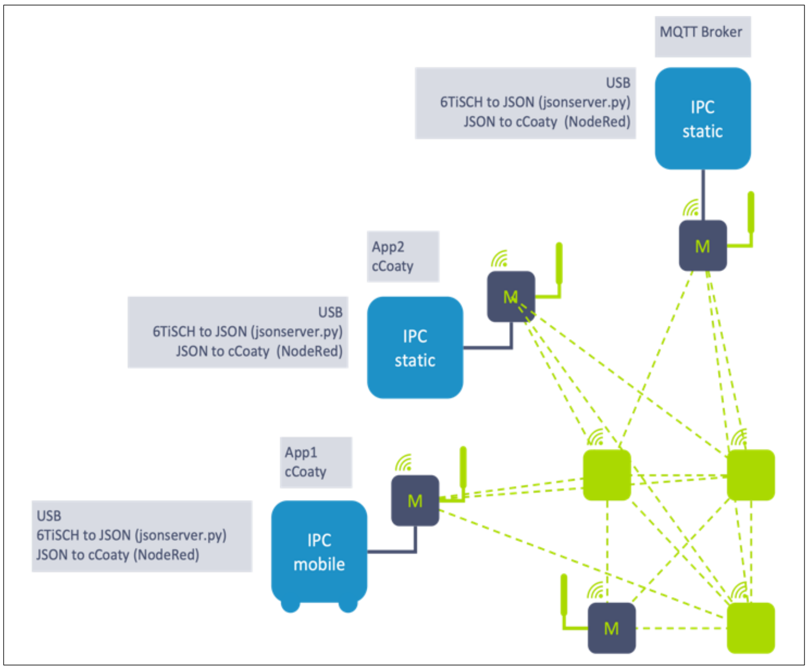

Figure 9: Test setup – Not possible DDA over SmartMeshIP / 6TiSCH network

Test setup 3 – Two SmartMesh IP / 6TiSCH cluster (AIOT) interact via DDA

Objective

Bring DDA into play initially, in the simplest way possible.

1. Implement and execute the DDA demo scenarios with the IPCs.
2. Demonstrate an end-to-end scenario, using data, events and actions from the nodes in the SmartMesh IP network, e.g. the DDA agent on the IPC A is subscribed to data from network B. The DDA agent on the IPC B publishes the data from network B, means:
   - gather some topics from a sensor*, *** node in cluster B
   - IPC B get the 6TiSCH data via USB, in JSON format, converts these to gRPC description, publish it via an DDA action
   - IPC A is subscribed to the DDA action and get the data**, ***

*) Use virtual sensor values / self-generated sensor values.

App#1 simulates a temperature sensor. Publishes temperature values at an interval of t seconds, from a minimum value of l to a maximum value of h with an increment/decrement of d.

App#2 simulates a brightness and motion sensor. Publishes brightness values at an interval of t seconds, from a minimum value of l to a maximum value of h with an increment/decrement of d.

App#3 simulates a location sensor. Publishes coordinate (x, y) values at an interval of t seconds, from a minimum value of lx and ly to a maximum value of hx and hy with an increment / decrement of dx an dy.

**) Use virtual actuators / dashboard for action visualization.

App#4 visualizes available topics as number, t/y chart (time series), x/y chart (map).

App#5 offers a configuration interface to modify application parameters (t, l, h, d) of app#1, 2 and 3.

***) Make use of topics needed for the EHS use-cases.

3. Evaluate how long does it take until a value, collected in cluster B, is transferred to a) IPC A, to b) Node 1 in cluster A?

Setup

The setup consists of two IPCs. First IPC is connected, via USB, with a SmartMesh IP Network Manager (M) with SmartMesh IP cluster A, second with cluster B.

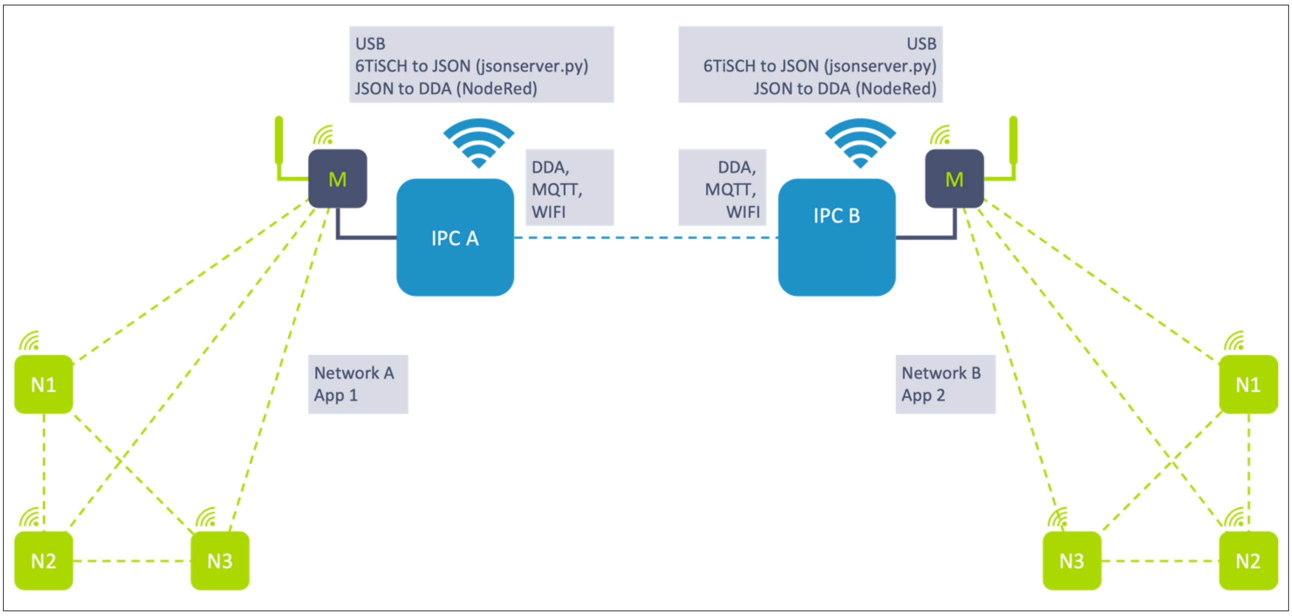

Figure 10: DDA setup

The IPCs are equipped with a DDA agent and communication layers.

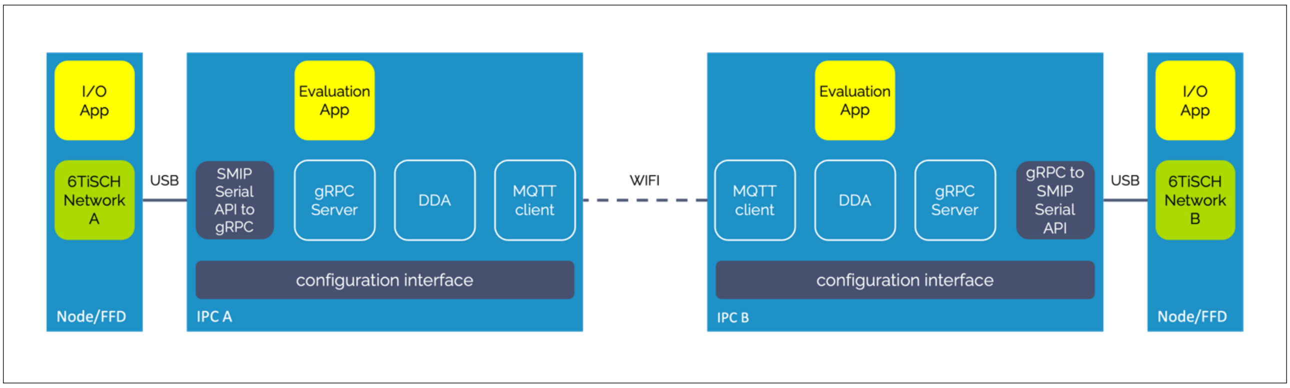

Figure 11: IPC communication stack for DDA setup

Result

- The setup was successfully deployed verifying the end-to-end DDA communication. However, it was identified that due to the IEEE 802.15.4 a package size of 127 Byte (subtracting headers 90 Bytes remaining) SmartMesh IP

/ 6TiSCH network will impose some communication limitations to the application overlay.

Discussion

- The impact of the payload limitations identified during the testing campaigns will have to be further assessed under the scope of the real payload specification of the different use cases, to understand whether SmartMesh IP / 6TiSCH network will be suitable for realizing the use case.

Test setup 4 – DDA on a SmartMesh IP / 6TiSCH stack

Test setup 4.1 – DDA on a SmartMesh IP / 6TiSCH (AIOT) stack

Objectives

Next step is the consideration of possible DDA agent adaptations to / integration in a SmarMesh IP network node, to clarify how a node can communicate with another via DDA pattern in a SmartMesh IP network.

Main goal is, that the constrained device (e.g. AIOT) act as a DDA client and exploit the DDA message types (Event, Action, Query).

Questions

- How can a SmartMesh IP node (AIOT) adapted to / integrated in an IPv6 / TCP / MQTT layer and at least DDA?

  In addition - get first experiences regarding:

    - Transfer speed and delays in transmissions.
    - Dependencies on data package sizes.
    - Connection establishment times.

Setup

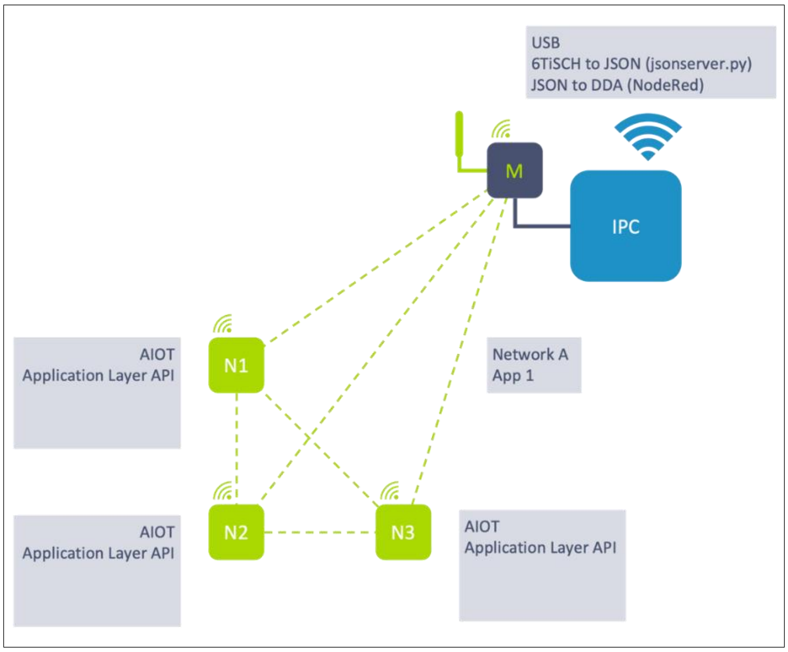

Figure 12: Test setup 4.1 – SmartMesh IP network DDA adaption

Result

Limitations due to closed source software of LTC 5800 chip from AD prevents to follow the constraint Coaty (DDA) integration.
As LTC 5800 proprietary stack requires the use of application layer API, direct transport layer communication, as needed to integrate AIOT nodes as DDA endpoints, is not possible.

Discussion

The following options were considered:

1) Design a Digital Twin. The IPC connected to the manager acts as the DDA endpoint and mirrors all operations to the actual node.
   Con: DDA will be terminated on a non-constrained device, thus need a protocol suited for constrained devices is not given anymore.
2) Proprietary API will be modified to extended to transparently tunnel TCP packets.
   Con: Significant efforts to be taken. Complicated stakeholder relations.
3) A TCP Tunnel will be implemented to transport TCP packets over the proprietary API to remote side (SLIP; text representation as payload to proprietary API).
   Con: Significant overhead leads to segmentation of single low payload messages to many individual WPAN messages to be transferred.
4) Use of open-source Stack (e.g. Contiki-NG, RIOT OS, OpenWSN, etc.) for access to transport layer and don't use the LTC5800 chip.
   Con: LTC5800 ("Dusty") has much better networking functionality compared to other radios such as for example on nRF*.
   *) https://crystalfree.atlassian.net/wiki/spaces/AIOT/pages/2193489991/Lab+18.+Meet+the+AIOT+Play
5) Implement DDA (on NRF of AIOT) using open-source stack down to SLIP (analog to 4). Transport SLIP encoded packages to LTC5800 and let LTC5800 take care of the lower layers.

Decision and proposed options to be selected is 4. This option is most close to constrained DDA use case and can be implemented in-house while providing performance insights.

Test setup 4.2 – DDA on a SmartMesh IP / 6TiSCH (OSS) stack

Objectives

Investigation of possible DDA adaptation according to option 4 from test setup 4.

- Integration in open-source stack Contiki-NG
- Consideration of access to transport layer (for example) on nRF and don't use the LTC5800 chip.
- Integration of TCP and MQTT client.

Setup

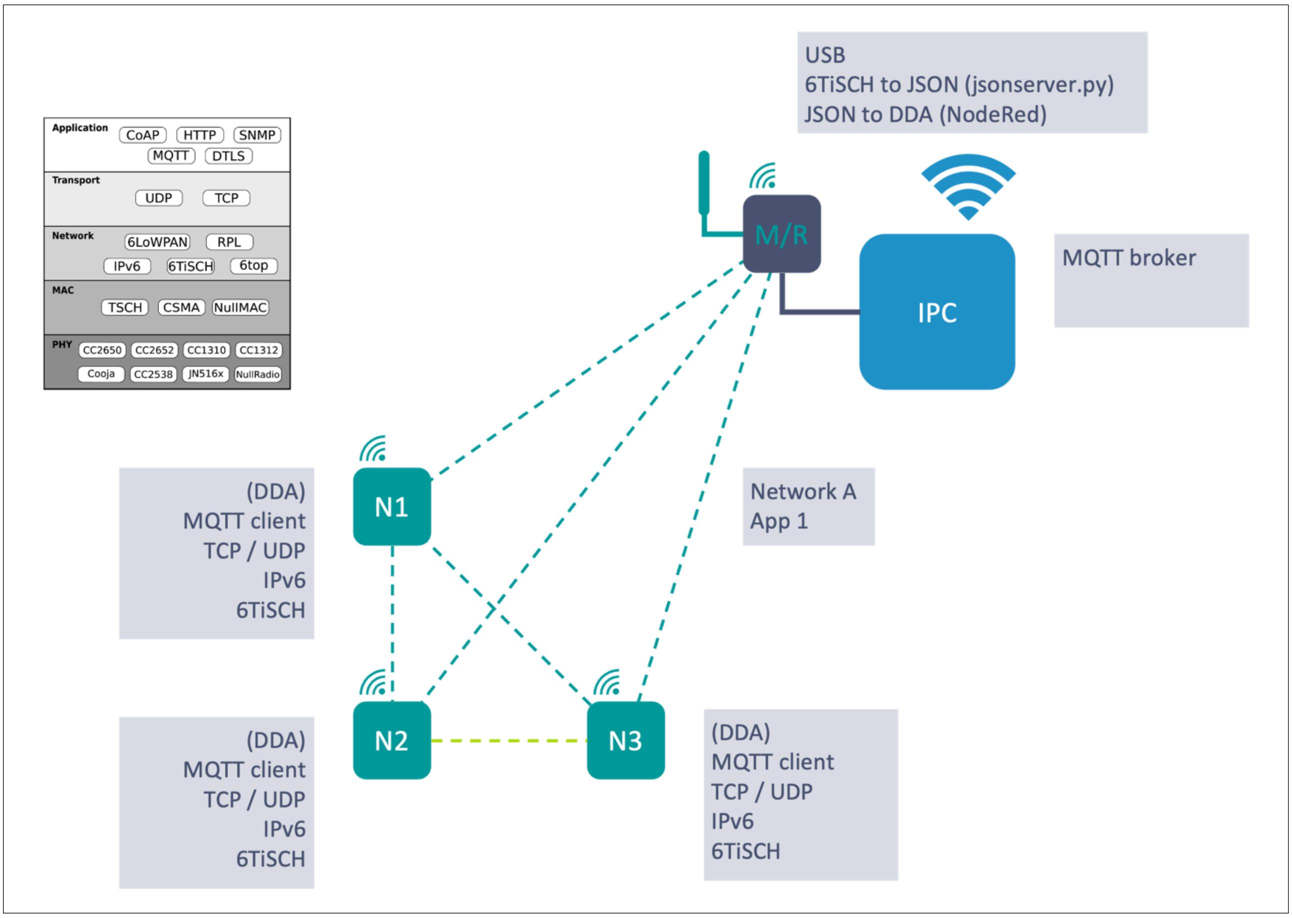

Figure 13: Test setup 4.2 – DDA communication in a 6TiSCH network

Approach

1. Use the Contiki-NG stack in simulation mode on a PC and setup an IPv6 - TCP connection. Integrate the MQTT client and setup a MQTT communication.
2. Installation of the Contiki-NG stack on a nFR board and setup an IPv6 - TCP connection. Integrate the MQTT client and setup a MQTT communication.

Result

- On the IPv6 network layer in the Contiki-NG environment (on the nRF board), the transport layer TCP and the MQTT client can be installed and operated. With the availability of the MQTT client, the prerequisite for the use of a DDA agent is basically met.
- IPv6 Routing is transparent for TCP and MQTT.

Discussion

- Due to there is no Go environment available for Contiki-NG, the present DDA implementation cannot be integrated. Possible solutions to work around this limitation are:
    - DDA needs to be ported into a programming language suitable for constraint devices.
    - Switch to a more powerful hardware, which also provides a nRF radio connection and a software runtime, which provides a go environment.

Test setup 5 – IPv6 routing demonstrator

Objectives

Use of IP-Routing between 6TiSCH and WIFI network to connect MQTT client in the network A to MQTT broker in the WIFI network.

Setup

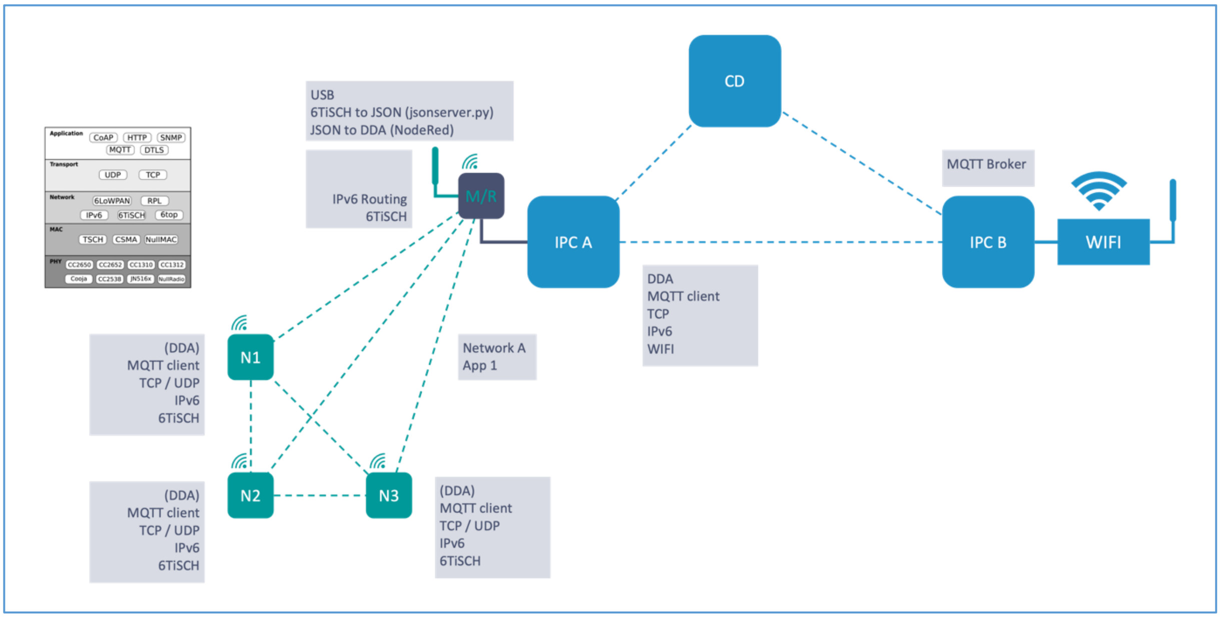

Figure 14: Test setup 5 – IPv6 routing

Result

- The MQTT client of the Contiki-NG stack in the 6TiSCH network connects to the MQTT broker in the WIFI network.

Discussion

- Installation of the DDA lightning demonstrator within this setup.
- The DDA can be used on the IPC without any restrictions. On the Contiki-NG nodes the MQTT clients can be used and statically configured for the lamp functionality to demonstrate the lightning scenario in this heterogeneous network setup.

Demonstrator setup

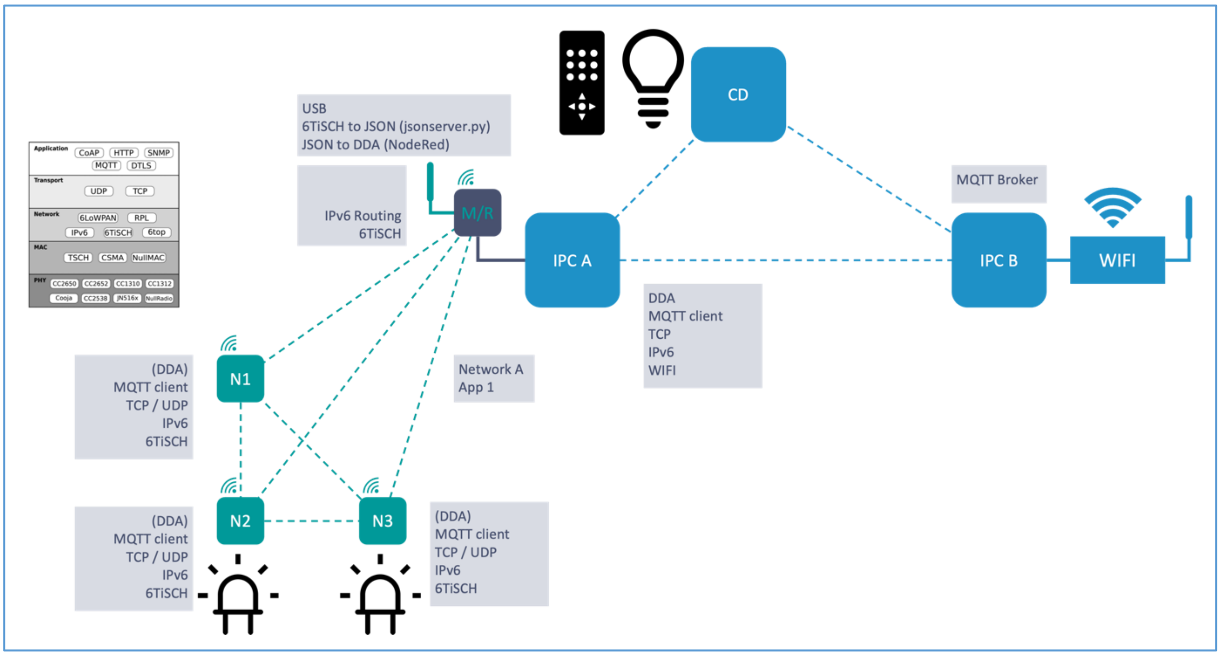

Figure 15: Test setup 5 – DDA lightning demonstrator

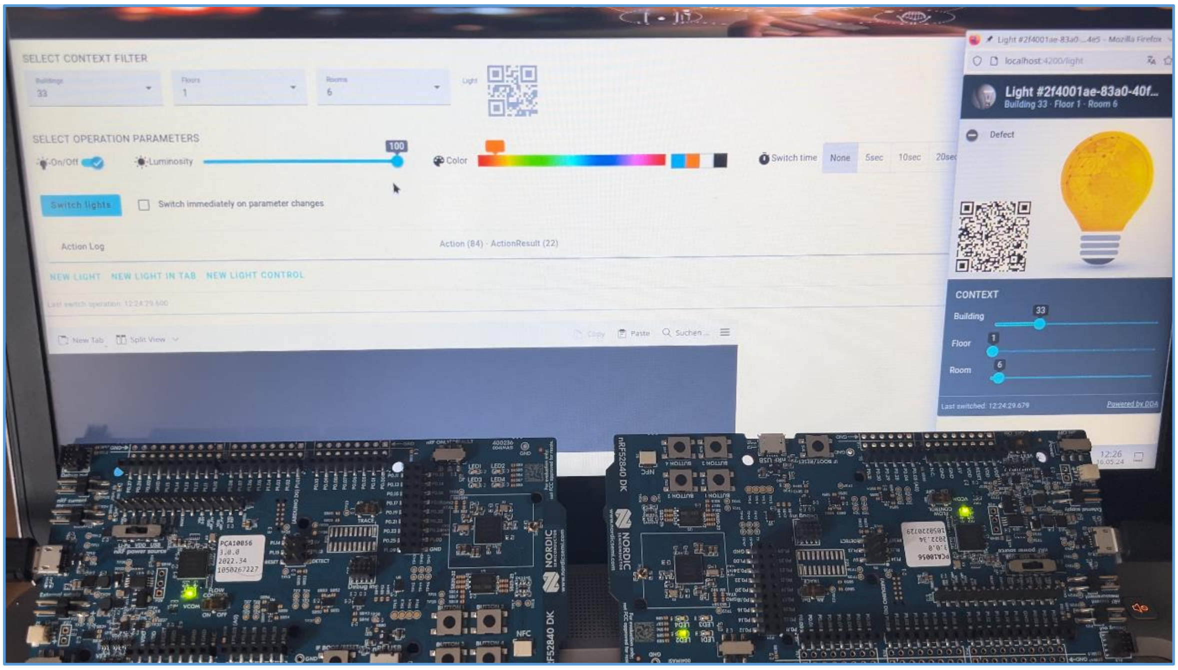

Figure 16: Test setup 5 – Physical structure of the DDA lightning demonstrator

A fully featured DDA (regarding reconfigurability and dynamic operability) on constrained devices only makes sense if the device can be reconfigured at runtime – which is not the case with constraint devices. In other words, for using DDA communication on constrained devices in a DDA network, an application-specific implementation is sufficient. Dynamic reconfiguration is neither practical nor necessary due to the fixed functionality of the devices.

Lessons Learned

Throughout the section we have aimed at answering the question of whether DDA-based communication can be supported in a heterogeneous network (6TiSCH, WLAN) of heterogeneous (constraint and fully featured, industrial and consumer devices), volatile devices.

Considering some boundary conditions, e.g. transmission power, etc., it was shown that SmartMesh IP nodes can be introduced into a network ad hoc and within acceptable time windows in order to interact with other nodes.

By integrating a DDA agent into a constraint device as part of the Lightning demonstrator, it was shown that constraint devices can be integrated into and used in a DDA network using IPv6 routing. These devices are then usually equipped with a completely dedicated function, without configurable variants, that are used precisely for this purpose.

Stil, during the realization of the proof-of-concepts scenarios of WP6 we will further assess the extent to which the current DDA implementation is applicable to constraint devices in such heterogeneous applications and real-world applications. One of the key focuses in this subsequent phase would be on the currently available and required communication patterns of DDA.

Conclusion

This deliverable constitutes a reference point for the utilisation of data-centric application overlay networks within swarm environments. It provides a solid theoretical foundation for data-centric middleware and demonstrates its value in meeting the OpenSwarm requirements, which are representative of various swarm use cases and are generally applicable to swarm applications. These foundations were supplemented with an analysis of state-of-the-art solutions and an in-depth examination of the Coaty framework.

As a result of this analysis, the key aspects required to evolve the Coaty framework for use in constraints environments were identified, leading to a new release of the Coaty framework tailored specifically for such environments. The updated framework was validated in representative test setups, aligned with the expected requirements of the Proof-of-Concepts in WP6.

In conclusion, the successful adaptation of the Coaty framework for constrained environments marks an important step in realising efficient data-centric communication in swarm robotics, laying the groundwork for further advancements in both academic research and practical applications.

Bibliography

[1] A. Haas, A. Rossberg, D. L. Schuff, B. L. Titzer, M. Holman, D. Gohman, L. Wagner, A. Zakai und J. Bastien, „Bringing the web up to speed with WebAssembly," *Proceedings of the 38th ACM SIGPLAN Conference on Programming Language Design and Implementation,* p. 185–200, 2017.

[2] J. Kreps, N. Narkhede, J. Rao und others, „Kafka: A distributed messaging system for log processing," *Proceedings of the NetDB,* Bd. 11, Nr. 2011, pp. 1-7, 2011.

[3] A. Katsifodimos und S. Schelter, „Apache Flink: Stream Analytics at Scale," *2016 IEEE International Conference on Cloud Engineering Workshop (IC2EW),* p. 193, 2016.

[4] T. Lewandowski, D. Henze, M. Sauer, J. Nickles und B. Bruegge, „A Software Architecture to enable Self-Organizing, Collaborative IoT Ressource Networks," *2020 Fifth International Conference on Fog and Mobile Edge Computing (FMEC),* pp. 70-77, 2020.

[5] J. Benet, „IPFS-content addressed, versioned, P2P file system (DRAFT 3)," *arXiv preprint arXiv:1407.3561,* pp. 1-11, 2014.

[6] M. Arutyunyan, A. Berestovskyy, A. Bratschi-Kaye, U. Degenbaev, M. Drijvers, I. El-Ashi, S. Kaestle, R. Kashitsyn, M. Kot, Y.-A. Pignolet, R. Rumenov, D. Sarlis, A. Sinpalean, A. Uta, B. Warinschi und A. Zapuc, „Decentralized and Stateful Serverless Computing on the Internet Computer Blockchain," *2023 USENIX Annual Technical Conference (USENIX ATC 23),* pp. 329-343, 7 2023.

[7] G. Bartolomeo, M. Yosofie, S. Bäurle, O. Haluszczynski, N. Mohan und J. Ott, „Oakestra: A Lightweight Hierarchical Orchestration Framework for Edge Computing," *2023 USENIX Annual Technical Conference (USENIX ATC 23),* pp. 215-231, 7 2023.

[8] G. Pardo-Castellote, „OMG Data-Distribution Service: architectural overview," *23rd International Conference on Distributed Computing Systems Workshops, 2003. Proceedings.,* pp. 200-206, 2003.

[9] L. Zhang, A. Afanasyev, J. Burke, V. Jacobson, k. claffy, P. Crowley, C. Papadopoulos, L. Wang und B. Zhang, „Named data networking," *SIGCOMM Comput. Commun. Rev.,* Bd. 44, Nr. 3, pp. 66-73, 2014.

[10] A. Corsaro, L. Cominardi, O. Hecart, G. Baldoni, J. E. P. Avital, J. Loudet, C. Guimares, M. Ilyin und D. Bannov, „Zenoh: Unifying Communication, Storage and Computation from the Cloud to the Microcontroller," *2023 26th Euromicro Conference on Digital System Design (DSD),* pp. 422-428, 2023.

[11] M. Shahrad, R. Fonseca, I. Goiri, G. Chaudhry, P. Batum, J. Cooke, E. Laureano, C. Tresness, M. Russinovich und R. Bianchini, „Serverless in the Wild: Characterizing and Optimizing the Serverless Workload at a Large Cloud Provider," *2020 USENIX Annual Technical Conference (USENIX ATC 20),* pp. 205-218, 2020.

Glossary

AGV	Automated Guided Vehicle
AMR	Autonomous Mobile Robot
API	Application Programming Interface
CI	Continuous Integration
DSO	Distribution System Operator
EC	Energy Community
ECO	Energy Community Operator
EHS	Environmental protection, Health management and Safety aspects
GNSS	Global Navigation Satellite System
IMU	Inertial Measurement Unit
JSON	JavaScript Object Notation – An open standard file format and data interchange format that uses human-readable text to store and transmit data objects.
LiDAR	Light Detection and Ranging
MANET	Mobile Ad Hoc Network
PMA	Protected Marine Environment
PoC	Proof of Concept
QoS	Quality of Service
REST	Representational State Transfer – A software architectural style
RTK	Real-Time Kinematics
ROS	Robot Operating System
SLAM	Simultaneous Localization and Mapping
UAV	Unmanned Aerial Vehicle

List of Figures