TaRDIS

D6.3: Report on the final iteration of TaRDIS toolbox components

Includes references to and descriptions of software prototypes.

Revision: v.1.0

Work package	WP6
Task	T6.1, T6.2, and T6.3
Due date	31/01/2026
Submission date	23/03/2026
Deliverable lead	NOVA
Version	1.0
Authors	João Leitão (NOVA), Nuno Preguiça (NOVA), Miloš Simić (UNS), Diogo Jesus (NOVA), João Brilha (NOVA), João Bordalo (NOVA), Dimitra Tsigkari (TID), Tomás Galvão (NOVA), Rafael Matos (NOVA), Felipe Carmos (NOVA), André Rijo (NOVA), Carla Ferreira (NOVA), João Gonçalo Pereira (NOVA), Diogo Paulico (NOVA), Rafael Costa (NOVA), Tamara Ranković (UNS)
Reviewers	Carla Ferreira (NOVA)
Abstract	This deliverable reports on the final iteration of the WP6 toolbox components developed within the TaRDIS project. It presents the complete set of technical contributions addressing decentralised communication, coordination, data management, observability, and execution across heterogeneous devices — from server-class machines to mobile phones and resource-constrained

Grant Agreement No.: 101093006
Call: HORIZON-CL4-2022-DATA-01
Topic: HORIZON-CL4-2022-DATA-01-03
Type of action: HORIZON- RIA

	embedded hardware. At the foundation of WP6 lies the Babel ecosystem, a protocol-centric runtime and programming model comprising Babel-Swarm for autonomic swarm-scale deployments, Babel-Android for mobile devices, and Micro-Babel for constrained embedded platforms. On top of this substrate, WP6 delivers interoperable building blocks for decentralised membership management, gossip-based dissemination, replicated data storage, application-level streaming, telemetry and monitoring, and secure coordination. The deliverable provides detailed technical descriptions and experimental evaluations for each contribution, including a large-scale validation with 5,000 concurrent nodes demonstrating 99.49% average reliability under emulated Internet-scale conditions. Cross-cutting integration aspects, a comprehensive assessment of all WP6 KPIs and requirements, and the evolution of results since Deliverables D6.1 and D6.2 are discussed. The deliverable further reports on the exploitation and sustainability planning conducted through the Horizon Results Booster programme, including the characterisation of the Babel Ecosystem as a Key Exploitable Result, the formulation of a unique value proposition, an exploitation roadmap with milestones and cost projections, and a structured risk assessment. Lessons learned from technical development, integration, and the transition from research artefacts to exploitable products are presented, alongside an outlook on future research and development directions centred on the consolidation of the ecosystem into Babel 2.
Keywords	Decentralised and Swarm systems, protocol composition, event-driven runtime, swarm systems, IoT, gossip protocols, membership management, CRDT-based replication, edge computing, embedded systems, fault tolerance, scalability, autonomic management, secure communication, telemetry, Babel framework, exploitation, open-source ecosystem.

Document Revision History

Version	Date	Description of change	List of contributor(s)
V0.1	13/01/2026	Defined structure of the deliverable	João Leitão (NOVA)
V1.0	21/03/2026	Complete Draft	João Leitão (NOVA)

Disclaimer

Funded by the European Union

Funded by the European Union (TARDIS, 101093006). Views and opinions expressed are however those of the author(s) only and do not necessarily reflect those of the European Union. Neither the European Union nor the granting authority can be held responsible for them.

COPYRIGHT NOTICE

Project funded by the European Commission in the Horizon Europe Programme		
Nature of the deliverable:	**R + DEM**	
Dissemination Level		
PU	*Public, fully open, e.g. web (Deliverables flagged as public will be automatically published in CORDIS project's page)*	✔
SEN	*Sensitive, limited under the conditions of the Grant Agreement*	
Classified R-UE/ EU-R	*EU RESTRICTED under the Commission Decision No2015/ 444*	
Classified C-UE/ EU-C	*EU CONFIDENTIAL under the Commission Decision No2015/ 444*	
Classified S-UE/ EU-S	*EU SECRET under the Commission Decision No2015/ 444*	

* R: Document, report (excluding the periodic and final reports)

DEM: Demonstrator, pilot, prototype, plan designs

DEC: Websites, patents filing, press & media actions, videos, etc.

DATA: Data sets, microdata, etc.

DMP: Data management plan

ETHICS: Deliverables related to ethics issues.

SECURITY: Deliverables related to security issues

OTHER: Software, technical diagram, algorithms, models, etc.

EXECUTIVE SUMMARY

Overview of WP6 role in TaRDIS

Work Package 6 provides the foundational runtime, protocol, and data management substrate of the TaRDIS project. Its role is to supply the mechanisms through which decentralised entities discover each other, communicate, coordinate their behaviour, manage shared state, and observe their own operation — all without reliance on centralised infrastructure. WP6 is organised around three tasks addressing complementary aspects of this challenge: decentralised communication and coordination (T6.1), data management and replication (T6.2), and monitoring, telemetry, and autonomic management (T6.3). The results produced by WP6 serve as the building blocks upon which the contributions of other work packages depend and are designed to operate across the full device spectrum targeted by TaRDIS — from cloud servers and edge gateways to mobile phones and battery-powered microcontrollers.

Summary of individual WP6 technical contributions

The centrepiece of WP6 is the Babel ecosystem, a protocol-centric runtime and programming model in which distributed protocols are treated as first-class composable units interacting exclusively through asynchronous events. The ecosystem comprises three complementary variants: Babel-Swarm for general-purpose and swarm-scale deployments on the JVM, with extensions for self-configuration, cryptographic identity management, secure channels, and adaptive parameter control; Babel-Android for mobile devices, encapsulating the runtime within an Android foreground service to maintain persistent protocol operation despite the platform's lifecycle constraints; and Micro-Babel, a C-based implementation on FreeRTOS targeting resource-constrained embedded hardware such as ESP32 and RP2040 devices.

On top of this runtime layer, WP6 delivers a set of interoperable building blocks. Decentralised membership management is provided by HyParView and its secure, autonomic, and self-discovery variants, complemented by Cyclon for lightweight peer sampling, X-BOT for application-aware overlay optimisation, and Random Tour for decentralised network-size estimation. Reliable message dissemination is supported by eager gossip broadcast, flood broadcast, one-hop broadcast, and an anti-entropy recovery protocol, each with secure variants providing encryption and message signing. Replicated data management is addressed by three complementary systems: Nimbus, a fully decentralised CRDT-based key-value store with partial replication and access control; PotionDB, a transactional store with materialised views; and Arboreal, a hierarchical cloud-edge replication system with causal+ consistency. External integration is supported through adapters for Cassandra, Hyperledger Fabric, C3, and Engage, as well as the integration of the Actyx middleware and a REST/WebSocket API for language-agnostic access. Observability is addressed through a centralised telemetry pipeline (Prometheus, Grafana), protocol-level metrics instrumentation, decentralised telemetry aggregation via gossip, and an initial machine learning pipeline for self-management training. Additional contributions include decentralised pub-sub streaming, namespace-based reconfiguration, secure coordination protocols for

decentralised energy markets, and the oar-p2p experimental infrastructure for large-scale emulated deployments.

Integration, validation, and KPI highlights

WP6 contributions have been validated through a combination of component-level experimentation, cross-component integration, and integrative demonstrators. The TaRDIS Messaging App exercises the full protocol stack across servers, Raspberry Pi devices, Android phones, and ESP32-based IoT hardware, demonstrating sub-second message delivery in local deployments and sustained operation of mobile devices for over 48 hours on battery. The large-scale experiment, conducted on the NOVA/DI research cluster with 5,000 concurrent Babel-Swarm processes under emulated Internet-scale conditions, achieved 99.49% average reliability and approximately 1.2 seconds average delivery latency, providing strong evidence that the core protocols scale to the target defined in the TaRDIS project objectives. All KPIs assigned to WP6 have been achieved and all requirements have been addressed, with caveats transparently discussed — most notably, the full-stack deployment at 5,000 nodes remains to be demonstrated due to cluster memory constraints, and the consolidation of the Babel ecosystem into a unified Babel 2 codebase was initiated but not completed within the project period.

Outlook beyond TaRDIS

The Babel Ecosystem has been identified as the primary Key Exploitable Result of WP6 and has been the subject of a structured exploitation programme through the Horizon Results Booster, resulting in a refined characterisation, a unique value proposition targeting IoT solution providers, an exploitation roadmap with milestones and financial projections, and a comprehensive risk assessment. The planned exploitation pathway includes open-source release of the unified core, creation of a spin-off company, IP protection through patent filing and trademark registration, pilot deployments in IoT and domotics, and community building. Early commercial adoption is already underway, with the Babel and Micro-Babel frameworks being used by small companies in Portugal for industrial IoT innovation, including a commercial product under development by ParadigmShift. The immediate technical priority beyond the project is the consolidation into Babel 2, converging on Java 17 as the unified base version, simplifying the security API towards secure-by-default communication, and developing structured cross-tier bridging abstractions to reduce the effort required for sensor-to-gateway integration. The combination of technical maturity, experimental validation, structured exploitation planning, and early industrial uptake positions the WP6 results for sustained impact beyond the conclusion of TaRDIS.

Funded by the European Union

Table of Contents

Funded by the European Union

1. Introduction

1.1 Scope and purpose

This deliverable reports on the final iteration of the work carried out within Work Package 6 (WP6) of the TaRDIS project. WP6 addresses the design, development, and experimental evaluation of fundamental mechanisms for communication, coordination, data management, and observability in decentralised and distributed systems operating under highly dynamic and heterogeneous conditions.

This deliverable (D6.3) builds upon the results previously reported in Deliverables D6.1 and D6.2, consolidating the work performed throughout the project and documenting the final state of the technical contributions developed within WP6. In line with the iterative reporting strategy adopted in TaRDIS, this deliverable does not aim to restate concepts, designs, or results already described in earlier deliverables, except where necessary to contextualise subsequent developments or to clarify how specific components have evolved. Instead, the focus of D6.3 is on presenting the complete set of new and final WP6 contributions as they stand at the end of the project, including extensions and refinements introduced after the delivery of D6.2, together with a systematic discussion of their experimental validation, limitations, and relevance with respect to the original project objectives and goals of this work package in the context of TaRDIS.

As the final deliverable of WP6, this document also provides a critical view of the work carried out during the project, highlighting the main technical challenges encountered, the design choices adopted to address them, and the lessons learned from both successful and less successful approaches, which derive from the fact that this technical work package was guided by significant research efforts. Where appropriate, it also discusses directions for further research and development that extend beyond TaRDIS's temporal scope.

1.2 WP6 objectives in the TaRDIS vision

TaRDIS research and innovation activities focus on the development of decentralised data-driven systems capable of operating in environments characterised by scale, heterogeneity, partial trust, and continuous change. The project targets scenarios in which computation, data, and decision-making are increasingly pushed towards the edge of the network, often involving large numbers of autonomous devices and components that must cooperate without relying on stable connectivity or centralised control to ensure continuous operation, correctness, and efficiency.

Within this broad context, WP6 is responsible for addressing a set of foundational problems that cut across multiple application domains and system configurations. These problems include how decentralised entities discover and organise themselves, how they exchange information efficiently and reliably, how shared or replicated state can be maintained under failures and potentially adversarial conditions, and how the behaviour of such systems can be observed and reasoned about at runtime.

Rather than focusing on a single architectural pattern or deployment model, WP6 adopts a deliberately flexible and high-level perspective. The mechanisms developed in this work package are designed to support a wide range of decentralised execution environments, from resource-rich edge nodes to highly constrained embedded devices, and from relatively stable overlays to highly dynamic and intermittently connected networks. As will become apparent throughout the document, these requirements emerged not only from the needs and requirements of TaRDIS's core use cases but also from a holistic approach to supporting current and future swarm systems. A central objective of WP6 is therefore to identify abstractions and design principles that remain applicable across these diverse settings, while still allowing concrete implementations to be specialised to particular constraints and requirements.

The results of WP6 form the technical basis on which other parts of the TaRDIS project are built. They are exploited by higher-level components developed in other work packages and are validated and evaluated both in isolation and, in different forms, within the project's use cases. At the same time, WP6 maintains a clear focus on producing reusable and extensible building blocks, intended to remain applicable beyond the specific scenarios explored within TaRDIS.

1.3 Reading guide and contribution-centric organisation

The structure of this deliverable reflects the diversity of technical contributions developed within WP6, while emphasising their integration within a coherent overall design. Each contribution reported in this document addresses a specific aspect of decentralised system support, such as communication, membership management, data replication, streaming, monitoring, or execution on constrained devices. Although these contributions are interrelated and often build on common abstractions, they are presented individually to provide a clear and precise account of their motivation, design, and evaluation. We note that the contributions reported here add to those presented in previous deliverables of the work package.

To ensure consistency and for the benefit of the reader, all WP6 contributions are described, as much as possible, using a common internal structure in this document. For each contribution, this deliverable aims to provide the problem context and motivating challenges; the objectives and design goals that guided the work; the main technical aspects of the proposed solution; its relationship with other WP6 components; and the ways in which it advances the state of the art. Whenever experimental results are available, these are reported and discussed, together with an explicit analysis of limitations and lessons learned.

In addition to the detailed presentation of individual contributions, the deliverable includes sections dedicated to cross-cutting aspects, such as system-level integration, demonstrators, and alignment with validation activities in other work packages. A separate section assesses the extent to which WP6 objectives and key performance indicators have been achieved, based on the evidence collected throughout the project.

Readers interested in an overview of WP6 results and their maturity may begin with Section 3, which summarises the main contributions and discusses their advances beyond the state of the art. Readers seeking detailed technical descriptions should refer

to Section 4, while Sections 5 and 6 provide a broader view on integration and validation. The final sections reflect on the outcomes of WP6, including lessons learned, exploitation potential, and directions for future work.

Funded by the European Union

2. WP6 Reporting Framework and Traceability

2.1 WP6 tasks and objectives

The work carried out within WP6 was organised around three closely related tasks, each addressing a complementary aspect of the support required by decentralised data-driven systems. While these tasks define the formal structure of the work package, they were not executed in isolation. Instead, they evolved in parallel, with frequent interactions and feedback loops, reflecting the intrinsic interdependencies between communication, coordination, data management, and observability in decentralised environments.

Task 6.1 (T6.1) focused on the design and evolution of mechanisms supporting decentralised communication and coordination. This task addressed fundamental challenges in how distributed entities discover one another, establish and maintain communication relationships, and coordinate their behaviour in the absence of centralised control. The work carried out under T6.1 laid the foundations for several higher-level contributions reported in this deliverable, including support for dynamic membership, protocol composition, and secure coordination patterns.

Task 6.2 (T6.2) addressed challenges related to data management in decentralised settings, with particular emphasis on replication, consistency, and resilience under failures and potentially adverse conditions (e.g., the presence of malicious actors in the system). The contributions developed within this task explore how shared or replicated state can be maintained across large numbers of nodes, potentially operating at the network edge, while accounting for failures, churn, and, in some cases, Byzantine behaviour. The results of T6.2 complement those of T6.1 by addressing the management of the state that is exchanged and coordinated through the communication mechanisms provided by the latter.

Task 6.3 (T6.3) focused on providing runtime support for the monitoring and management of complex swarm systems, with the overarching goal of enabling autonomic behaviour in decentralised applications that operate at scale, under churn, and across heterogeneous devices and administrative domains. The starting point of this task is the observation that manual, human-centric management rapidly becomes impractical as the size and diversity of a swarm grow, and that, even when feasible, it tends to be error-prone and carry non-negligible operational risk. Consequently, T6.3 addresses two tightly coupled challenges. The first is the systematic acquisition, processing, and dissemination of telemetry data describing the runtime behaviour of swarm applications and their underlying protocol compositions, including resource consumption at the device level, performance indicators at the process and application levels, and protocol-specific metrics necessary to interpret the emergent behaviour of decentralised executions. The second challenge concerns the ability to coordinate and execute reconfiguration actions across a potentially large number of distributed components, possibly deployed across different administrative domains, in a way that remains scalable and avoids single points of failure.

In this context, the work carried out under T6.3 has been driven by the need to expose telemetry and control hooks in a form compatible with runtime adaptation strategies, including machine-learning-based ones. In particular, T6.3 has a strong interaction with

Funded by the European Union

the project activities on decentralised intelligence, since defining reconfiguration plans for complex decentralised systems is difficult to capture with static rules or domain-specific heuristics alone and instead benefits from approaches that learn from observed system behaviour and adjust operational parameters accordingly. At the same time, this interaction raises additional requirements that are specific to decentralised settings, namely that monitoring and reconfiguration mechanisms should avoid relying on a single central entity, and should instead support designs in which different segments of the system can carry out local control actions with limited coordination, preserving scalability while still enabling coherent system-wide behaviour when required.

Orthogonal to these tasks, WP6 also had the complementary goal of supporting the integration of existing technologies for use in the design of new and emerging swarm applications. This goal was pursued during the early execution of the Project and was reported previously in D6.1, mostly in the context of the Babel framework. It should, however, be noted that in the work produced at the last stage of the project, we have also integrated and leveraged technologies available for small integrated devices (e.g., Free RTOS), which also enables interoperability between TaRDIS components and that ecosystem. We, however, do not focus our presentation on this aspect.

Although each task has a distinct focus, the technical results reported in this deliverable often span multiple tasks. For this reason, the organisation of D6.3 does not follow a strict task-by-task presentation. Instead, tasks are used as a reporting and traceability mechanism, while the main body of the deliverable is organised around individual technical contributions, each of which may relate to one or more WP6 tasks.

2.2 Mapping between WP6 tasks, contributions, and sections

Although WP6 is formally structured around three tasks, the technical contributions reported in this deliverable cut across task boundaries systematically and intentionally. This reflects that decentralised systems cannot be cleanly decomposed along a single dimension, such as communication, data management, or monitoring, without introducing artificial separations that do not align with how such systems are designed or evaluated in practice. As a result, each contribution reported in Section 4 is associated with a primary WP6 task that motivated its development and often also contributes to the objectives of one or more additional tasks.

The work reported under Section 4.1, which concerns the Babel ecosystem as a substrate for decentralised communication and coordination, originates primarily from Task T6.1. This contribution addresses the fundamental abstractions and runtime mechanisms required to structure decentralised interactions and provides the basis upon which several other WP6 contributions are built. Its role is therefore foundational with respect to the remainder of the work package.

Section 4.2 reports on Micro-Babel, which extends the communication and coordination mechanisms developed under T6.1 to highly constrained and embedded execution environments. While the core abstractions follow directly from the work carried out in T6.1, this contribution also responds to requirements that emerged from T6.3, namely the need to observe and manage decentralised systems that span devices with widely differing resource profiles.

The decentralised membership mechanisms presented in Section 4.3 are primarily associated with Task T6.1, as they address the problem of maintaining coherent views of participation and connectivity in dynamic environments. At the same time, these mechanisms are a prerequisite for several data management and monitoring contributions developed under T6.2 and T6.3, which rely on membership information to scope replication, dissemination, and aggregation activities. It should be noted that this contribution to the work package was motivated by the satellite used case put forward by GMV.

Section 4.4, which covers advanced replicated data management at the edge, is primarily based on Task T6.2. This contribution focuses on maintaining shared state in the presence of failures and adversarial conditions and builds directly on the communication and coordination mechanisms developed under T6.1. Its design and evaluation also inform the requirements for monitoring and management mechanisms developed under T6.3, particularly with respect to the observability of replication behaviour and fault conditions.

The decentralised application-level streaming mechanisms described in Section 4.5 are primarily associated with Task T6.3. While they rely on the communication substrate developed under T6.1, their main contribution lies in supporting efficient, scalable dissemination of information streams and exposing runtime behaviour relevant to adaptive management strategies.

Sections 4.6 and 4.7 report on telemetry, monitoring, aggregation, and experimental infrastructure, and are closely aligned with the objectives of Task T6.3. These contributions address the need to observe, analyse, and reason about the runtime behaviour of decentralised systems, as well as to support the experimental evaluation of WP6 mechanisms under realistic conditions. They also serve as enabling infrastructure for validating contributions developed under T6.1 and T6.2.

Finally, Section 4.8, which addresses secure and privacy-aware decentralised support for market-like coordination scenarios, is primarily associated with Task T6.1, as it explores coordination protocols and interaction patterns built on top of the WP6 communication substrate. At the same time, this contribution informs the objectives of T6.3 by highlighting the requirements that security- and privacy-sensitive coordination places on monitoring and reconfiguration mechanisms, being specifically motivated by the renewable energy decentralized markets use case put forward by EDP.

By making these relationships explicit, this deliverable preserves a clear correspondence between the formal task structure of WP6, and the concrete technical results produced during the project, while at the same time providing an accurate account of the cross-cutting nature of the work. This mapping also serves as the basis for the assessment of WP6 objectives and key performance indicators presented later in the document.

2.3 Validation and evidence methodology

The validation of the technical contributions developed within WP6 reflects both the diversity of the problems addressed and the exploratory nature of several research activities conducted during the project. Rather than relying on a single, uniform validation

methodology, WP6 adopts a contribution-specific approach, in which evaluation strategies are tailored to the objectives and assumptions of each result produced by the project.

For contributions concerned with communication, coordination, and data management mechanisms, validation typically involves experimental evaluation of prototype implementations. These evaluations focus on aspects such as scalability, fault tolerance, overhead, and responsiveness, and are conducted under controlled conditions that enable systematic exploration of system behaviour across varying workloads, network conditions, and failure scenarios. In several cases, these experiments rely on emulation environments specifically developed or extended within WP6, enabling reproducible experimentation at scales that would be difficult to achieve in physical deployments, in line with the project's overall goals to demonstrate the applicability of technology developed in TaRDIS to systems of up to 5.000 devices. Scaling experiments to this number of devices was itself a challenge faced and addressed in the context of WP6.

Contributions developed under T6.3 require a complementary validation perspective. In this case, the primary concern is not only the performance of individual mechanisms, but also their suitability as runtime support for monitoring and management in complex decentralised systems. Validation, therefore, focuses on the ability to expose relevant telemetry with acceptably low overhead, to aggregate and disseminate this information in a scalable manner, and to support the execution of coordinated reconfiguration actions without introducing central points of control or failure. Where applicable, these aspects are evaluated through experiments that combine telemetry acquisition, aggregation, and control actions, illustrating how monitoring data can drive adaptation at runtime.

For contributions that explore new design spaces or propose novel abstractions, validation is necessarily limited to feasibility studies and qualitative analysis. In these cases, the limitations of the evaluation are explicitly acknowledged, and the reported results are framed in terms of insights gained and trade-offs identified, rather than definitive performance claims. This approach is consistent with the research-oriented nature of WP6 and with the role of the work package in advancing the state of the art while informing future research and development efforts.

In addition to isolated experimental evaluation, several WP6 contributions are validated through their integration with other components developed within the project and through their use in demonstrators and use cases reported in WP7. This form of validation provides evidence of interoperability and practical applicability, even when exhaustive performance evaluation is not feasible or does not represent the intended deployment scenarios. Additionally, WP6 has devoted some additional effort in developing its own case study application, based on a messaging application that can execute across servers, small devices, mobile phones, and even small integrated devices, that exploit the technical contributions of the work package to demonstrate an applicational scenario where heterogeneous devices interact and can affect their environment through the use of demonstrative IoT devices (sensors and actuators). We discuss the design of this demonstrator in Section 6.

Throughout this deliverable, validation evidence is reported alongside the corresponding technical contributions and is used as the basis for the assessment of WP6 objectives and key performance indicators presented later in the document. By explicitly relating evaluation results to the underlying design goals and assumptions, the deliverable aims to provide a transparent and technically grounded account of the extent to which WP6 has achieved its intended outcomes.

Funded by the European Union

3. WP6 Toolbox Overview and Maturity Summary

3.1 Overview of WP6 technical contributions

The work carried out within WP6 resulted in a set of inter-connected technical contributions that together address several of the foundational challenges faced by decentralised and heterogeneous swarm systems. These contributions span multiple layers of system support, ranging from communication and coordination mechanisms to data management, monitoring, and execution on constrained devices. While each contribution targets a specific problem space, they are designed to operate cohesively and to be combined as needed to support the development, deployment, and operation of complex decentralised applications.

From an architectural perspective, the WP6 contributions constitute a core part of what is referred to throughout the project as the TaRDIS Toolbox. In this context, the term toolbox should not be understood as a monolithic platform or a fixed software stack, but rather as a collection of interoperable mechanisms, abstractions, and runtime components that can be composed to address different system requirements. This perspective is well aligned with TaRDIS's overall strategy, which emphasizes modularity, protocol composition, and the separation between generic mechanisms and application-specific logic.

At the foundation of WP6 lies the Babel ecosystem, which provides a runtime and programming model for developing decentralised protocols and applications. Babel's contribution is not limited to the implementation of individual protocols, but rather to the definition of a common execution model and API that allows protocols for membership, dissemination, replication, and coordination to be developed, composed, and reused in a uniform way. This addresses a limitation observed in much of the existing literature and tooling, where decentralised protocols are typically presented as isolated artefacts, often tied to specific implementations or assumptions, making their reuse and integration difficult[1]. The Babel ecosystem, as evolved from the original version of Babel[2], in the context of TaRDIS, provides the substrate - specially tailored for heterogeneous swarm applications - upon which most other WP6 contributions are built.

Building on this foundation, WP6 includes contributions that extend decentralised communication and coordination support to settings characterised by high dynamism and partial connectivity. This includes membership abstractions designed to cope with intermittent links and evolving communication opportunities, motivated by scenarios such as satellite swarms and other mobile systems. These contributions revisit classical

[1] J. Leitão, J. Pereira, and L. Rodrigues. "Epidemic Broadcast Trees." In *Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007)*, Beijing, China, October 2007, pp. 301–310. IEEE Computer Society.

[2] P. Fouto, P. Á. Costa, N. Preguiça N, and J. Leitão. "Babel: A framework for developing performant and dependable distributed protocols." In *Proceedings of the* 41st International Symposium on Reliable Distributed Systems (SRDS) 2022 Sep 19 (pp. 146-155). IEEE Computer Society.

decentralised membership protocols, such as HyParView[3] and Cyclon[4], and explore alternative designs that explicitly account for constrained and time-varying communication patterns, rather than assuming a relatively stable underlying network[5].

WP6 also addresses data management challenges that arise when decentralised applications require access to shared or replicated state across large numbers of nodes, potentially deployed at the network edge. The contributions in this area focus on replication and consistency mechanisms that are compatible with decentralised operation and that avoid the scalability and availability limitations of strongly consistent systems in geo-distributed and edge environments. In particular, WP6 explores designs based on causal and causal+ consistency[6], extending existing approaches by supporting partial and dynamic replication, fault tolerance, and deployment beyond traditional data centre settings. This positions the work relative to existing causal+ systems such as COPS, Orbe[7], and GentleRain[8] [9], which were not designed with edge deployments or highly dynamic replica sets in mind.

Complementing communication and data management, WP6 includes a set of contributions focused on the observability and management of decentralised systems at runtime. These contributions are motivated by the need to move beyond centralised monitoring approaches, which introduce bottlenecks and single points of failure, and towards decentralised telemetry acquisition, aggregation, and interpretation mechanisms that remain effective under churn and partial failures. The previous WP6 deliverables highlighted the importance of integrating fine-grained metric collection into the communication substrate itself, enabling uniform observation of protocol- and application-level behaviour. The work reported in this deliverable extends this direction by consolidating telemetry support, aggregation mechanisms, and interfaces for runtime reconfiguration, with the explicit goal of enabling autonomic management strategies.

---

[3] J. Leitão, J. Pereira, and L. Rodrigues. “HyParView: A Membership Protocol for Reliable Gossip-Based Broadcast.” In *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2007)*, Edinburgh, United Kingdom, June 2007, pp. 419–429. IEEE Computer Society.

[4] S. Voulgaris, D. Gavidia, and M. van Steen. “CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays.” *Journal of Network and Systems Management*, vol. 13, no. 2, June 2005, pp. 197–217.

[5] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulié. “SCAMP: Peer-to-Peer Lightweight Membership Service for Large-Scale Group Communication.” In *Proceedings of the 3rd International Workshop on Networked Group Communication (NGC 2001)*, London, UK, November 2001, pp. 44–55. Springer.

[6] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen. “Don’t Settle for Eventual: Scalable Causal Consistency for Wide-Area Storage with COPS.” In *Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP 2011)*, Cascais, Portugal, October 2011, pp. 401–416. ACM.

[7] S. Duan, S. B. O’Neil, S. P. Mu, and J. C. Corbett. “Orbe: Scalable Causal Consistency Using Dependency Matrices and Physical Clocks.” In *Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2015)*, Broomfield, CO, USA, October 2015, pp. 401–416. USENIX Association.

[8] P. Bailis, A. Fekete, M. J. Franklin, A. Ghodsi, J. M. Hellerstein, and I. Stoica. “Coordination Avoidance in Database Systems.” *Proceedings of the VLDB Endowment*, vol. 8, no. 3, November 2014, pp. 185–196.

[9] P. Bailis, A. Fekete, M. J. Franklin, A. Ghodsi, J. M. Hellerstein, and I. Stoica. “Scalable Causal Consistency with GentleRain.” In *Proceedings of the 39th International Conference on Very Large Data Bases (VLDB 2013)*, Trento, Italy, August 2013, pp. 468–479.

Another important venue for WP6 technical contributions concerns support for decentralised execution across heterogeneous devices, including highly constrained embedded platforms. The Micro-Babel contribution extends the abstractions and design principles developed in the Babel ecosystem to environments where resources such as memory, processing power, and energy are severely limited. This work reflects the growing importance of integrating edge and embedded devices into decentralised systems and ensures that the mechanisms developed within WP6 are not restricted to resource-rich execution environments.

Finally, WP6 includes contributions that explore secure and privacy-aware coordination patterns in decentralised settings, motivated by scenarios such as decentralised energy markets and other forms of resource coordination. Rather than focusing on application-specific logic, these contributions investigate how coordination protocols can be designed to minimise trust assumptions, limit information disclosure, and operate without central authorities. In doing so, they highlight requirements and trade-offs that influence the design of the underlying communication, monitoring, and management mechanisms developed elsewhere in WP6.

In terms of maturity, the contributions reported in this deliverable span a range of development stages. Some components, such as the Babel ecosystem and several communication and data management mechanisms, have been iteratively developed, implemented, and experimentally evaluated across multiple project iterations, and are integrated into demonstrators and use cases reported in WP7. Other contributions are more exploratory in nature, focusing on feasibility, design space exploration, and identifying trade-offs rather than on exhaustive validation. This diversity reflects both the research-oriented nature of WP6 and its role in advancing the state of the art while providing concrete artefacts that can be exploited within and beyond the TaRDIS project.

The remainder of this deliverable builds on this overview by first synthesising the advances beyond the state of the art achieved by WP6 as a whole, and then providing detailed, contribution-specific descriptions and evaluations in Section 4.

3.2 Advances beyond the State-of-the-Art

The technical contributions developed within WP6 address a set of challenges that have been extensively studied in the literature on distributed and decentralised systems. These problems include membership management in large-scale, dynamic networks; reliable dissemination of information without central coordination; replication and consistency of shared state; monitoring and management of distributed executions; and the integration of heterogeneous devices into coherent system architectures. Over the past two decades, a substantial body of work has proposed solutions for each of these aspects, often under specific assumptions regarding network stability, trust, or deployment environment.

In the area of decentralised communication and membership management, gossip-based and epidemic protocols have long been recognised as effective mechanisms for

achieving scalability and robustness in large-scale systems. Protocols such as SCAMP[10] and Cyclon[11] introduced lightweight peer sampling techniques that allow nodes to maintain partial, continuously evolving views of the system, thereby enabling probabilistic guarantees for connectivity and dissemination efficiency. HyParView[12] further refined these ideas by explicitly separating active and passive views to improve reliability under churn, particularly for gossip-based broadcast, a solution still employed today in practical systems, such as the protocol support provided by Iroh[13].

While these protocols provide strong foundations for decentralised cooperation and communication, they are typically presented as self-contained solutions, tailored to specific dissemination or membership objectives. Less attention has been paid to the problem of how such protocols can be systematically integrated, reused, and composed within larger systems that combine multiple decentralised mechanisms. Moreover, many existing approaches implicitly assume relatively homogeneous execution environments and comparatively stable connectivity patterns, assumptions that are increasingly violated in modern swarm and edge-based systems.

WP6 advances beyond this state of the art by adopting a protocol-oriented runtime perspective, which is exploited in the design and development of the Babel ecosystem. Rather than introducing yet another gossip or membership protocol, Babel provides a common execution model and programming interface that enables the development, composition, and coordinated execution of decentralised protocols addressing different concerns. This shifts the focus from individual protocol designs to the problem of structuring decentralised systems as compositions of interacting protocols, a dimension largely absent from prior work on epidemic dissemination and membership management[14].

In the domain of replicated data management, the literature and state of the art have extensively explored trade-offs between consistency, availability, and performance. Strongly consistent systems, while providing intuitive semantics, suffer from scalability and latency limitations in geo-distributed and failure-prone environments. This has led to the development of systems that are eventually and causally consistent, such as

[10] A. J. Ganesh, A.-M. Kermarrec, and L. Massoulié. “SCAMP: Peer-to-Peer Lightweight Membership Service for Large-Scale Group Communication.” In Proceedings of the 3rd International Workshop on Networked Group Communication (NGC 2001), London, UK, November 2001, pp. 44–55. Springer.
[11] S. Voulgaris, D. Gavidia, and M. van Steen. “CYCLON: Inexpensive Membership Management for Unstructured P2P Overlays.” Journal of Network and Systems Management, vol. 13, no. 2, June 2005, pp. 197–217.
[12] J. Leitão, J. Pereira, and L. Rodrigues. “HyParView: A Membership Protocol for Reliable Gossip-Based Broadcast.” In Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2007), Edinburgh, United Kingdom, June 2007, pp. 419–429. IEEE Computer Society.
[13] https://www.iroh.computer
[14] J. Leitão, J. Pereira, and L. Rodrigues. “Epidemic Broadcast Trees.” In Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007), Beijing, China, October 2007, pp. 301–310. IEEE Computer Society.

COPS[15], Orbe[16], and GentleRain[17], which explicitly leverage designs that relax consistency guarantees to achieve better scalability and availability.

These systems, however, were primarily designed for data centre or cloud-based deployments, with relatively stable replica sets and well-provisioned nodes. WP6 advances beyond the state of the art by exploring replication and consistency mechanisms explicitly designed for decentralised and edge-centric environments, where replica membership may change dynamically, nodes may be resource-constrained, and failures may include not only crashes but also Byzantine behaviour. In doing so, WP6 bridges the gap between work on causal consistency and research on fault tolerance and decentralised coordination, extending existing models to contexts that were not originally considered.

Observability and runtime management constitute another area where the existing state of the art exhibits significant limitations when applied to decentralised systems. Traditional monitoring solutions typically rely on centralised collection and aggregation of metrics, which introduces scalability bottlenecks and single points of failure. Prior work has explored decentralised monitoring and aggregation techniques, including epidemic and tree-based approaches[18], to address these issues. Nevertheless, such approaches are often developed independently from the communication and coordination substrates on which the monitored systems are built.

WP6 advances beyond these approaches by tightly integrating telemetry collection, aggregation, and dissemination mechanisms into the same runtime framework used for protocol execution and coordination, an idea that has already been explored in wireless ad hoc systems[19]. This integration enables a more coherent view of system behaviour, in which protocol-level and application-level metrics can be interpreted in relation to each other, and monitoring data can be directly exploited to drive runtime adaptation and reconfiguration. This perspective aligns with the goals of autonomic computing but extends them to decentralised settings where central controllers are undesirable or infeasible.

Finally, the literature on decentralised systems has traditionally focused on relatively homogeneous execution environments, often abstracting away differences between servers, mobile devices, and embedded platforms. With the increasing importance of

[15] W. Lloyd, M. J. Freedman, M. Kaminsky, and D. G. Andersen. “Don’t Settle for Eventual: Scalable Causal Consistency for Wide-Area Storage with COPS.” In Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP 2011), Cascais, Portugal, October 2011, pp. 401–416. ACM.
[16] S. Duan, S. B. O’Neil, S. P. Mu, and J. C. Corbett. “Orbe: Scalable Causal Consistency Using Dependency Matrices and Physical Clocks.” In Proceedings of the 12th USENIX Symposium on Operating Systems Design and Implementation (OSDI 2015), Broomfield, CO, USA, October 2015, pp. 401–416. USENIX Association.
[17] P. Bailis, A. Fekete, M. J. Franklin, A. Ghodsi, J. M. Hellerstein, and I. Stoica. “Scalable Causal Consistency with GentleRain.” In Proceedings of the 39th International Conference on Very Large Data Bases (VLDB 2013), Trento, Italy, August 2013, pp. 468–479.
[18] J. Leitao, L. Rosa and L. Rodrigues, "Large-Scale Peer-to-Peer Autonomic Monitoring," 2008 IEEE Globecom Workshops, New Orleans, LA, USA, 2008, pp. 1-5, doi: 10.1109/GLOCOMW.2008.ECP.18.
[19] P. Á. Costa and J. Leitao, "Practical Continuous Aggregation in Wireless Edge Environments," in 2018 IEEE 37th Symposium on Reliable Distributed Systems (SRDS), Salvador, Brazil, 2018, pp. 41-50, doi: 10.1109/SRDS.2018.00015.

edge and IoT deployments, this assumption is no longer tenable. WP6 advances the state of the art by explicitly addressing execution across heterogeneous devices, including highly constrained embedded systems, while preserving a common programming and coordination model. The Micro-Babel contribution exemplifies this direction, demonstrating how decentralised protocol abstractions can be adapted to environments with strict resource constraints without abandoning interoperability with more capable nodes.

Taken together, the advances achieved within WP6 do not consist of incremental improvements to individual protocols but rather of a coherent rethinking of how decentralised communication, data management, monitoring, and execution support can be structured to address the requirements of modern swarm and edge-based systems. The detailed contributions presented in Section 4 build upon this positioning by providing concrete designs, implementations, and evaluations that substantiate these advances beyond the state of the art.

3.3 Maturity and validation status of contributions

The technical contributions reported in this deliverable exhibit varying levels of maturity and validation, reflecting both the diversity of challenges addressed and the abstractions developed within the context of WP6, as well as the research-oriented nature of several activities carried out during the project. Rather than converging on a single artefact or implementation, WP6 pursued a strategy of incremental development, evaluation, and integration of multiple fundamental building blocks and components, each targeting a specific aspect of decentralised and swarm system support. As a result, maturity should be understood not only in terms of implementation completeness, but also in terms of conceptual stability, integration with other WP6 components, and the extent to which a contribution has been exercised under realistic experimental conditions.

Some of the core contributions of WP6, most notably the Babel ecosystem and the associated communication and coordination abstractions, have reached a relatively advanced level of maturity, and these were explored for commercial viability in the Booster program (discussed in Section 8.2). These components have been iteratively refined across multiple project phases and deliverables, starting with earlier versions of Babel (prior to the start of the TaRDIS project) and evolving through D6.1 and D6.2 to their current state. They are supported by concrete implementations, have been integrated with several other WP6 contributions, and have been used as a foundation for the demonstrators and use cases reported by the project (namely, in this deliverable and the final deliverables of Work Package 7). Their validation includes experimental evaluation under controlled conditions, as well as indirect validation through reuse and integration in more complex application-level scenarios.

A similar level of maturity is evident across several contributions addressing decentralised data management and replication. These contributions are grounded in well-established consistency models and extend them to more dynamic and heterogeneous environments. Prototype implementations have been developed and evaluated using large-scale emulation-based experimental setups, allowing the exploration of scalability, fault tolerance, and performance trade-offs under varying conditions. While these mechanisms are not intended to constitute production-ready

storage systems, they provide validated designs and empirical evidence supporting the feasibility and relevance of the proposed approaches for decentralised and edge-centric deployments.

Other contributions, particularly those related to observability, telemetry, and runtime management, are at an intermediate level of maturity. The mechanisms for metrics collection, aggregation, and dissemination are supported by working implementations and have been exercised in experimental settings, including integration with other WP6 components. Their validation focuses primarily on demonstrating scalability, robustness, and acceptable overhead, as well as on illustrating how telemetry can be exploited to support adaptation and reconfiguration. At the same time, these contributions intentionally leave several design choices regarding control logic and decision-making strategies open, influenced by application-specific requirements and, in some cases, by ongoing work on decentralised intelligence within the project.

Contributions addressing execution across heterogeneous and constrained devices, such as Micro-Babel, also exhibit a mixed maturity profile. On the one hand, these contributions provide concrete implementations that demonstrate the feasibility of deploying decentralised protocol abstractions on resource-constrained platforms, including small integrated devices. On the other hand, they explore a design space that is still evolving, particularly regarding trade-offs among functionality, resource usage, and interoperability. In this case, validation is therefore primarily concerned with feasibility, resource consumption, and integration with other WP6 components, rather than with large-scale performance evaluation. This said, this particular set of results has been integrated into the application-level demonstrator built in the context of WP6 operation.

Finally, some contributions reported in this deliverable are explicitly exploratory, particularly those investigating secure and privacy-aware coordination patterns in decentralised settings. These contributions are motivated by concrete use cases but primarily aim to explore protocol design alternatives and identify requirements and trade-offs that influence the broader WP6 architecture. Their validation relies on validated prototypes, emulations, and qualitative analysis, rather than on exhaustive experimental evaluation. This is consistent with their role in informing future research and guiding the evolution of the underlying communication and management mechanisms.

Overall, the maturity and validation status of WP6 contributions should be interpreted in light of the work package's objectives. WP6 was not intended to deliver a single integrated system at production readiness level, but rather to advance the state of the art by developing, evaluating, and integrating a set of foundational mechanisms for decentralised systems. The combination of mature components, validated prototypes, and exploratory designs reported in this deliverable reflects this objective and provides a solid basis for both further research and practical exploitation beyond the duration of the TaRDIS project. However, the core results of WP6 have reached a TRL level of 5 or 6, with fully functional prototypes being tested either on an emulated (and realistic) environment in the Lab or in relevant environments.

3.4 Evolution since D6.2

The work reported in this deliverable represents a consolidation and extension of the results previously presented in Deliverables D6.1 and D6.2, reflecting both the natural

progression of the research and development activities within WP6 and the integration of additional contributions developed during the final phase of the project. Rather than introducing a fundamentally new direction, D6.3 builds upon the architectural foundations, abstractions, and design principles established earlier, while extending them in scope, depth, and level of validation.

A first important aspect of this evolution concerns the consolidation of the communication and coordination substrate developed within WP6. While D6.2 already reported stable versions of the Babel ecosystem and its associated abstractions, the work carried out since then has focused on strengthening their integration with other WP6 components, clarifying their role as a unifying runtime for decentralised protocol execution, and extending their applicability to a broader range of execution environments. This includes both the refinement of existing mechanisms and the incorporation of lessons learned from experimental evaluation and demonstrator development.

A second major evolution relates to the expansion and maturation of contributions addressing execution on heterogeneous and constrained devices. Although support for resource-constrained environments was already identified as a relevant direction in D6.2, the work reported in this deliverable introduces Micro-Babel as a concrete instantiation of this vision. This contribution extends the WP6 abstractions to embedded platforms, demonstrating interoperability with other Babel-based components and validating the feasibility of deploying decentralised protocol logic across devices with widely differing resource profiles. This evolution reflects a shift from conceptual discussion to concrete implementation and integration.

Significant progress has also been made in data management and replication mechanisms. While D6.2 presented initial designs and experimental results for decentralised replication under relaxed consistency models, D6.3 reports extended implementations and evaluations that consider larger scales, more dynamic replica sets, and more adverse conditions. In particular, the work reported here strengthens the connection between replication mechanisms and the communication and membership abstractions provided by WP6, resulting in designs that are better aligned with the requirements of edge-centric and swarm-based deployments.

Another notable evolution concerns support for observability, telemetry, and runtime management. In D6.2, these aspects were introduced as enabling mechanisms for monitoring decentralised executions and supporting adaptation. Since then, this line of work has been extended and consolidated, resulting in a more coherent set of mechanisms for metrics collection, aggregation, and dissemination, as well as clearer interfaces for triggering and coordinating reconfiguration actions. This evolution reflects a deeper engagement with the autonomic management objectives originally outlined for WP6 and provides a more concrete basis for integrating telemetry-driven adaptation strategies.

In addition to extending existing contributions, D6.3 also reports several new or substantially expanded results that were not fully developed at the time of D6.2. These include decentralised application-level streaming mechanisms, advanced telemetry aggregation strategies, and experimental infrastructure for large-scale and realistic evaluation of decentralised systems. Although some of these contributions are

exploratory, they address gaps identified in earlier phases of the project and provide valuable insights into the design space for decentralised system support.

Finally, D6.3 reflects an evolution in the way WP6 results are presented and contextualised. Compared to D6.2, this deliverable adopts a more contribution-centric organisation, explicitly positioning each result with respect to the state of the art, systematically discussing validation and limitations, and clarifying how individual contributions relate to the broader objectives of WP6 and the TaRDIS project. This evolution is intended to provide a clearer and more comprehensive account of the work carried out within WP6, and to support both technical assessment and future exploitation of the reported results.

Funded by the European Union

4. WP6 Technical Contributions

This section presents the set of technical contributions developed within WP6 in a detailed and systematic manner. Each contribution addresses a specific aspect of decentralised and swarm system support, ranging from communication and coordination mechanisms to data management, observability, execution on constrained devices, and secure coordination patterns. While these contributions are inter-connected and often built upon shared abstractions and runtime support, they are presented individually to provide a clear account of their motivation, design choices, and evaluation.

The organisation of this section reflects the contribution-centric approach adopted throughout this deliverable. Rather than structuring the presentation strictly along task boundaries, each contribution is discussed as a self-contained result, explicitly contextualised within WP6's objectives and the broader goals of the TaRDIS project. This approach allows the reader to understand both the specific problems addressed by each contribution and how they integrate into a coherent technical substrate.

For consistency and clarity, all contributions are described following a common internal structure. Each subsection discusses the problem context and motivation; the objectives and design goals; the main technical aspects of the solution; its relationship with other WP6 components; its advances beyond the state of the art; the available experimental validation; and the limitations and lessons learned. This structure is intended to support both focused reading of individual contributions and cross-cutting analysis across the work package.

The section begins with the Babel ecosystem, which provides the foundational runtime and programming model upon which many of the WP6 contributions were built. Subsequent sections progressively address more specialised abstractions, mechanisms, and extensions, including support for constrained devices, advanced membership management, data replication, streaming, monitoring, and secure coordination.

4.1 Babel Ecosystem

As discussed above, the Babel ecosystem constitutes the foundational substrate of WP6. Within the scope of TaRDIS, Babel has evolved into two primary and complementary variants, each addressing a distinct but interrelated execution context. Babel-Swarm targets large-scale, dynamic, and autonomic swarm systems operating on general-purpose computing platforms, while Babel-Android extends the same programming and execution model to mobile devices, explicitly addressing the constraints imposed by the Android runtime and application lifecycle. Together, these variants form the core of the Babel ecosystem in TaRDIS and provide the basis for several other WP6 contributions.

4.1.1 Context and Motivation

The design and implementation of decentralised and distributed systems traditionally rely on a combination of ad hoc protocol implementations, middleware libraries, and application-specific coordination logic. In much of the existing literature, decentralised

protocols are introduced as standalone artefacts, evaluated in isolation, and often tightly coupled to specific assumptions about the execution environment, communication model, or application semantics. This approach makes it difficult to reuse protocols across systems, to compose multiple protocols within the same execution, or to evolve systems incrementally as requirements change.

The Babel framework was originally proposed[20] to address this gap by providing a dedicated runtime and programming model for distributed and decentralised protocols, rather than for applications directly. Its motivation stems from the observation that many distributed systems can be understood as compositions of interacting protocols, each responsible for a specific concern such as membership management, message dissemination, replication, or failure handling. However, existing development approaches provide limited support for explicitly expressing such compositions, reasoning about their interactions, or experimenting with alternative protocol designs in a controlled manner.

Prior to TaRDIS, Babel had already been used to prototype and evaluate a range of decentralised protocols, demonstrating the feasibility and usefulness of a protocol-centric development approach. The context of TaRDIS introduced additional challenges that further motivated the evolution of Babel. These include the need to support highly dynamic swarm environments, heterogeneous deployments spanning cloud, edge, and embedded devices, tighter integration with monitoring and management mechanisms, and the ability to evolve and reconfigure protocol compositions at runtime. WP6 builds on the original Babel ideas to address these challenges systematically.

4.1.2 Objective and Design Goals

Within WP6, the primary objective of the Babel ecosystem is to serve as a reusable and extensible substrate for the development, composition, and execution of decentralised protocols in complex, heterogeneous, and evolving environments. Rather than providing a fixed set of services or abstractions, Babel aims to offer a minimal yet expressive execution model that closely aligns with how decentralised protocols are described and reasoned about in the research literature.

A key design goal is the clear separation between protocol logic and low-level system concerns such as networking, concurrency, and timer management. By abstracting these aspects behind a well-defined runtime API, Babel allows protocol designers to focus on correctness and behaviour, while still retaining control over relevant performance and interaction properties. Another important goal is to support explicit protocol composition, enabling multiple protocols to coexist, interact, and coordinate within the same system without being hard-wired together.

In the context of TaRDIS, additional goals emerged, including support for runtime observability, integration with decentralised monitoring and management mechanisms, and adaptability to heterogeneous execution environments. These goals influenced the

[20] P. Fouto, P. Á. Costa, N. Preguiça N, and J. Leitão. “Babel: A framework for developing performant and dependable distributed protocols.” In *Proceedings of the* 41st International Symposium on Reliable Distributed Systems (SRDS) 2022 Sep 19 (pp. 146-155). IEEE Computer Society.

evolution of Babel variants developed or extended during the project, such as Babel-Swarm and Babel-Android, while preserving compatibility with the original design philosophy.

4.1.3 Technical Description

Overall Architecture and Execution Model

The Babel ecosystem is organised around a protocol-centric runtime architecture in which distributed protocols are treated as the primary units of composition and execution. Rather than structuring systems around services or application components, Babel adopts an execution model in which each protocol encapsulates a specific aspect of distributed behaviour and interacts with the environment exclusively through well-defined events.

Protocols execute within a single runtime instance and are scheduled cooperatively by Babel's core. The runtime is responsible for serialising event delivery, invoking protocol handlers, managing timers, and coordinating interactions between protocols. This design avoids exposing low-level concurrency mechanisms to protocol developers, while still enabling highly concurrent behaviour at the system level. The resulting execution model closely mirrors the event-driven structure commonly used to describe decentralised algorithms in the literature, thereby reducing the semantic gap between specification and implementation.

Event Model and Protocol Interaction

Protocols in Babel respond to a small set of well-defined event types, including network message arrivals, timer expirations, and internal notifications from other protocols. Each protocol explicitly declares the events it is interested in, making dependencies between protocols visible and analysable.

Inter-protocol interaction is realised through asynchronous event exchange rather than direct method calls or shared state. This decoupling is a deliberate design choice that supports modularity and composability. Protocols can be added, removed, or replaced - at development time - without requiring changes to other components, provided that the event interfaces they expose remain compatible. This interaction model also enables cross-cutting functionality, such as monitoring or security, to be introduced as independent protocols that observe or influence others' behaviour.

Communication Abstractions and Channels

Communication between nodes is mediated through abstract channels that encapsulate transport-specific concerns. Channels provide a uniform interface for sending and receiving messages, while hiding details such as connection establishment, message framing, and error handling. Multiple channel implementations can coexist within the same runtime, enabling protocols to select communication semantics appropriate to their needs.

This abstraction is particularly important in heterogeneous deployments, where different nodes may rely on different transport technologies or networking stacks. By decoupling protocol logic from transport details, Babel enables the same protocol implementation to

be reused across environments, including cloud deployments, edge nodes, and mobile devices, provided that suitable channel implementations are available.

Protocol Lifecycle and Dynamic Reconfiguration

Protocols in Babel have an explicit lifecycle, which includes initialisation and activation phases. This lifecycle management allows protocols to be dynamically instantiated and configured at runtime, supporting incremental system evolution and adaptation. Parameters influencing protocol behaviour can be supplied at initialisation time or updated dynamically, enabling the system to react to changes in topology, workload, or environmental conditions.

In the context of TaRDIS, this lifecycle support was extended to accommodate autonomic behaviour. Protocol parameters can be exposed as reconfigurable elements, enabling external or internal control logic to adjust protocol behaviour in response to observed system metrics. This capability is central to the integration of monitoring and management mechanisms developed under Task T6.3.

Babel-Swarm: Autonomic and Secure Swarm Execution

Babel-Swarm extends the original Babel runtime with mechanisms specifically designed to support large-scale and highly dynamic swarm systems. A central addition is systematic support for self-configuration. Nodes can join a swarm without relying on preconfigured contact points, using discovery mechanisms integrated into the runtime. Configuration parameters required by protocols can be dynamically obtained from other nodes or external sources, reducing manual configuration effort and enabling deployment in ad-hoc environments.

Security is another major extension introduced in Babel-Swarm. Communication channels can be transparently secured using cryptographic mechanisms, and the runtime provides abstractions for managing node identities and cryptographic material. Rather than enforcing a single trust model, Babel-Swarm exposes flexible hooks that allow protocols and applications to define how identities are validated and how trust relationships are established. This approach reflects the diversity of security assumptions found in decentralised systems and avoids hard-coding a particular security policy into the runtime.

Babel-Swarm also introduces support for adaptive management. Protocol parameters can be marked as adaptive, allowing their values to be adjusted dynamically based on runtime observations. The runtime integrates lightweight mechanisms for estimating global properties, such as swarm size, and exposes these estimates to control logic that can trigger reconfiguration actions. This enables protocols to adapt their behaviour, for example by adjusting dissemination fanout or timeout values, in response to changing conditions.

Babel-Android: Mobile and Lifecycle-Aware Execution

Babel-Android adapts the Babel-Swarm execution model to the constraints of the Android platform. While Android applications are written in Java or Kotlin, their execution is governed by a strict lifecycle managed by the operating system, which can suspend or terminate background components to conserve resources. This poses a fundamental

challenge for decentralised protocols that rely on long-lived communication channels and periodic maintenance tasks.

To address this, Babel-Android encapsulates the Babel runtime in a foreground service, ensuring protocol execution and communication continue even when the user interface is not active. The runtime is decoupled from the application's UI logic, which interacts with Babel protocols through asynchronous requests and notifications. This separation allows mobile devices to participate fully in swarm executions while preserving the responsiveness and energy efficiency required by the Android ecosystem.

Additional adaptations were required to account for differences in available Java versions, networking APIs, and logging facilities. These adaptations preserve the core abstractions and execution model of Babel-Swarm, ensuring that protocols developed for desktop or server environments can be reused on Android devices with minimal modification.

4.1.4 Relationship with Other WP6 Components

Most WP6 contributions rely directly on Babel as their execution substrate. Membership protocols, data replication mechanisms, streaming primitives, telemetry collection, and reconfiguration support are implemented as Babel protocols or leverage its runtime abstractions. Micro-Babel reinterprets the same design principles for highly constrained devices, ensuring conceptual continuity across heterogeneous environments.

As a result, Babel acts as the primary integration point within WP6, enabling otherwise diverse contributions to interoperate within a common execution and interaction model.

4.1.5 Advances Beyond the State-of-the-Art

The problem of structuring the development and execution of distributed systems has been addressed by a variety of frameworks and middleware platforms. Systems such as Appia[21] and Cactus[22] explored protocol composition and configurable protocol stacks, primarily in the context of dependable distributed systems. Actor-based frameworks such as Erlang/OTP and Akka provide powerful abstractions for concurrent and distributed programming, but focus primarily on application-level structuring rather than on protocol-level composition[23].

More recent libraries, such as libp2p, offer modular networking components for peer-to-peer systems, but typically provide communication primitives and services rather than a protocol-centric execution model[24].

Babel advances beyond these approaches by explicitly targeting decentralised protocols as the primary unit of design and execution. Its runtime model closely mirrors the abstractions used in protocol specifications, enabling direct translation from design to

[21] H. Miranda, A. Pinto, and L. Rodrigues. "Appia: A Flexible Protocol Kernel Supporting Multiple Coordinated Channels." In Proceedings of the 21st International Conference on Distributed Computing Systems (ICDCS 2001), Phoenix, AZ, USA, April 2001, pp. 707–710. IEEE.
[22] R. C. van Renesse, K. P. Birman, and S. Maffeis. "Horus: A Flexible Group Communication System." Communications of the ACM, vol. 39, no. 4, 1996, pp. 76–83.
[23] J. Armstrong. "Making Reliable Distributed Systems in the Presence of Software Errors." PhD Thesis, Royal Institute of Technology (KTH), 2003.
[24] J. Benet. "IPFS – Content Addressed, Versioned, P2P File System." arXiv:1407.3561, 2014.

implementation. Unlike protocol stacks or actor frameworks, Babel provides explicit support for protocol composition, interaction, and lifecycle management within a unified runtime, while remaining agnostic to application semantics.

The evolution of Babel within WP6 further extends this positioning by integrating concerns such as observability, reconfiguration, and deployment heterogeneity directly into the protocol runtime. Rather than treating these aspects as external services, they are addressed as first-class concerns that influence protocol design and interaction. This combination of protocol-centric abstraction, modular composition, and runtime introspection is not well supported by existing frameworks and represents a significant advance in the systematic engineering of decentralised systems.

4.1.6 Experimental Validation and Evaluation

The experimental validation of the Babel ecosystem combines baseline evidence obtained prior to the TaRDIS project with TaRDIS-specific evaluation of the extensions introduced in WP6, most notably those realised in the Babel-Swarm variant. This section summarises the evaluation methodology and the main results, while referring the reader to Deliverable D6.2 for a detailed presentation of experimental data, plots, and quantitative analysis.

As discussed earlier, the original Babel framework predates TaRDIS, and its initial experimental evaluation focused on validating the feasibility of a protocol-centric runtime architecture. These baseline results, reported in the original Babel publication, demonstrated that the event-driven execution model and protocol composition mechanisms do not impose prohibitive overhead when compared to more monolithic protocol implementations. While these results are not considered TaRDIS outcomes, they provide the empirical foundation that motivated the adoption and subsequent evolution of Babel within WP6.

The TaRDIS-specific experimental evaluation focuses on the Babel-Swarm variant, which extends the original runtime with mechanisms to support large-scale, highly dynamic swarm systems. These extensions include automatic configuration, decentralised security hooks, runtime estimation of global properties, and adaptive reconfiguration support. The evaluation of Babel-Swarm was carried out and reported in detail in Deliverable D6.2; here, we summarise the experimental setup and the key observations derived from those experiments.

The experimental evaluation of Babel-Swarm was conducted in an emulation-based environment, where multiple nodes running the Babel-Swarm runtime were deployed under controlled conditions. Each node ran a representative protocol stack comprising membership management and dissemination protocols, augmented with Babel-Swarm runtime services. The experiments explored a range of swarm sizes by progressively increasing the number of participating nodes, enabling analysis of scalability properties as the system size grew.

The evaluation focused on metrics that are particularly relevant to swarm-oriented systems, including message overhead per node, convergence time after node joins, and the additional cost introduced by runtime services compared to a baseline execution without Babel-Swarm extensions. By contrasting executions with and without these

extensions enabled, the evaluation isolated the impact of automatic configuration, estimation, and adaptation mechanisms.

The results reported in D6.2 show that the additional runtime services introduced in Babel-Swarm incur a measurable but controlled overhead. While total communication activity naturally increases with the number of nodes in the system, the per-node overhead remains bounded and grows gradually with swarm size. This behaviour indicates that the Babel-Swarm extensions do not introduce central coordination points or bottlenecks that would compromise scalability. Moreover, the observed convergence times and stability properties demonstrate that automatic configuration and estimation mechanisms remain effective even as the system scales.

Taken together, these results provide evidence that the Babel-Swarm extensions introduced within WP6 successfully extend the original Babel runtime with autonomic and swarm-oriented capabilities, without fundamentally altering the scalability characteristics of protocol execution. They validate the feasibility of integrating self-configuration and adaptive behaviour at the runtime level and support the role of Babel-Swarm as a suitable substrate for the swarm-based scenarios considered in TaRDIS.

Further details on the experimental methodology, quantitative results, and comparative analysis can be found in Deliverable D6.2, which provides a comprehensive account of the Babel-Swarm evaluation.

4.1.7 Limitations and Lessons Learned

The experience gained through the design, evolution, and use of the Babel ecosystem within WP6 has highlighted several limitations, both in the original framework and in the variants developed or extended during the TaRDIS project. While many of these limitations were not immediately apparent in earlier iterations, their impact became increasingly evident as Babel was applied to more heterogeneous environments, integrated with a broader set of protocols, and exercised in realistic deployment scenarios.

One important limitation concerns the coexistence of multiple Babel codebases targeting different execution environments and Java versions. In particular, the evolution of Babel-Swarm and Babel-Android along partially divergent technical paths introduced additional development and maintenance effort, especially for protocol implementations intended for reuse across variants. Differences in supported Java versions and platform-specific constraints forced protocol developers to accommodate subtle incompatibilities, reducing the effectiveness of code reuse. This experience has motivated an ongoing effort to consolidate the Babel variants under a single, unified Java version and a more harmonised runtime core, with the explicit goal of reducing fragmentation and simplifying protocol development.

A second limitation relates to the lifecycle management of protocols. In the current Babel design, protocols are instantiated and activated at runtime but cannot be explicitly paused, suspended, or stopped once execution has begun. While this design choice simplified early implementations and avoided complex state management issues, it proved restrictive in scenarios where finer-grained control over protocol execution is required. This limitation is particularly relevant in mobile environments, such as those targeted by Babel-Android, where operating-system-imposed application lifecycle events

may require protocols to be temporarily suspended or gracefully resumed. Similar concerns also arise in user-facing systems running on laptops or personal devices, where resource availability and user interaction patterns may change dynamically. Addressing this limitation is a key aspect of the ongoing evolution of the Babel runtime.

The design of communication channels also revealed limitations as Babel was applied to more dynamic network environments. In earlier versions, channels were relatively intrusive and tightly coupled to underlying network assumptions, making it difficult to handle changes in network infrastructure transparently. For example, scenarios in which a device's IP address changes at runtime, due to mobility or network reconfiguration, were not adequately supported. This limitation affected both robustness and ease of deployment in highly dynamic settings. As a result, the new integrated version of Babel is revisiting the channel abstraction to better decouple protocol logic from network-level details and to provide more flexible support for dynamic connectivity.

Security mechanisms introduced in Babel-Swarm constitute another important source of lessons learned. While these mechanisms provided a flexible foundation for secure communication and identity management, they also imposed a significant burden on protocol developers. In practice, the security abstractions proved too intrusive from a programming perspective, requiring developers to reason explicitly about cryptographic material, trust relationships, and security policies at multiple points in protocol logic. This complexity hindered adoption and experimentation and influenced the decision not to aggressively explore application-level solutions based on these mechanisms within WP6. The experience gained in this regard has led to a rethinking of how security support should be exposed in the Babel ecosystem, with the aim of providing stronger security guarantees while significantly reducing the cognitive and implementation overhead for developers.

Taken together, these limitations highlight a recurring theme in the evolution of the Babel ecosystem: design decisions that are appropriate in controlled or homogeneous environments may become problematic when the system is pushed towards greater heterogeneity, dynamism, and scale. At the same time, the identification of these limitations has directly informed the ongoing redesign and integration efforts that aim to unify the Babel variants, simplify their programming model, and improve their suitability for modern decentralised and swarm-based systems. These lessons are therefore not merely retrospective but actively shape the future direction of the Babel ecosystem beyond the TaRDIS project.

4.2 Micro-Babel: Embedded and Constrained Devices Support

The contributions reported in this section address a fundamental limitation in the practical deployment of decentralised systems: the difficulty of extending decentralised communication and coordination mechanisms to highly constrained embedded devices. While the architectural principles introduced for the Babel ecosystem in Section 4.1 enable the construction of distributed/decentralised protocols on general-purpose platforms, many of the sensing, actuation, and interaction points relevant to TaRDIS scenarios reside on devices with limited computational resources, restricted memory, and heterogeneous networking capabilities.

Within WP6, Micro-Babel was developed to bridge this gap by providing a lightweight runtime that enables embedded devices to directly participate in decentralised protocol execution, rather than acting merely as passive data producers or gateways. This contribution primarily aligns with the objectives of Task 6.1 by extending decentralised communication and coordination abstractions to resource-constrained platforms, and supports Task 6.3 by enabling embedded-level observability, telemetry dissemination, and local decision-making under adverse network conditions.

Micro-Babel is not intended as a reimplementation of the full Babel ecosystem for embedded devices. Instead, it selectively reinterprets and adapts the core architectural principles of Babel - explicit protocol modularity, event-driven execution, and decentralised coordination - under the strict constraints imposed by microcontroller-based platforms. In doing so, it enables TaRDIS deployments to span heterogeneous systems, from embedded sensors and actuators to mobile and edge devices, while preserving a coherent decentralised execution model.

4.2.1 Context and Motivation

The design of Micro-Babel is motivated by the observation that most existing Internet-of-Things (IoT) and embedded system architectures remain fundamentally centralised, even when deployed in ostensibly distributed settings. Embedded devices typically depend on gateways, cloud services, or coordinator nodes for communication, coordination, and control. These assumptions break down in disaster scenarios and infrastructure-degraded environments, which are central to the TaRDIS use cases.

Micro-Babel addresses this limitation by enabling embedded devices to execute decentralised protocols locally, allowing them to discover peers, exchange messages, react to network changes, and coordinate behaviour without relying on external infrastructure. The primary objectives guiding this contribution are threefold.

First, Micro-Babel aims to preserve the protocol-centric programming model established by Babel, ensuring conceptual continuity across TaRDIS software artefacts. Protocols remain explicit entities that encapsulate behaviour and interact through well-defined events (including messages).

Second, the runtime is designed to support heterogeneous embedded platforms, including the Raspberry Pi Pico family and ESP32-based devices, which are representative of the hardware used in TaRDIS demonstrators. This objective requires careful separation between platform-independent runtime logic and platform-specific networking and concurrency mechanisms.

Third, Micro-Babel prioritises predictable resource usage and robustness. Memory allocation, event queues, and protocol registration are bounded and, where possible, statically configured, enabling long-running operations under constrained conditions and reducing the risk of runtime failures due to resource exhaustion.

4.2.2 Objective and Design Goals

Micro-Babel targets embedded devices that operate under the following conditions: limited RAM and flash memory, absence of managed runtimes or dynamic class loading, restricted concurrency support, and reliance on lightweight networking stacks or direct

radio interfaces. Devices may be intermittently connected, experience frequent topology changes, and operate without access to cloud services or central coordination points.

The runtime assumes that devices may support one or more communication technologies, such as Wi-Fi, BLE, or low-power radio links, and that network availability may change dynamically during execution. Rather than hiding these dynamics, Micro-Babel exposes connectivity changes explicitly to protocol logic through events, enabling decentralised adaptation strategies.

4.2.3 Technical Description

Runtime Architecture

Micro-Babel is structured as a lightweight runtime core composed of a small number of cooperating components, each responsible for a distinct aspect of decentralised execution. These components interact through explicit interfaces and shared data structures, avoiding implicit dependencies and dynamic coupling.

At the core of the runtime is an event-driven execution model. All protocol logic is triggered by events, which represent external stimuli (such as message arrivals or network changes) as well as internal triggers (such as timers or protocol requests). This design mirrors Babel's execution semantics while allowing tighter control over memory usage and scheduling.

A central design decision in the development of Micro-Babel was to build the embedded runtime on top of FreeRTOS[25]. FreeRTOS is a widely adopted real-time operating system for microcontrollers and resource-constrained platforms, designed specifically to provide predictable task scheduling, low overhead, and straightforward integration with diverse networking stacks and hardware peripherals. Its kernel implements a deterministic preemptive scheduler with bounded latency, enabling the predictable execution of time-critical components such as event dispatchers and protocol handlers. FreeRTOS achieves this with a minimal footprint, making it suitable for microcontroller hardware classes where RAM and flash resources are limited and non-deterministic runtime behaviour can undermine decentralised protocol correctness.

The choice of FreeRTOS is justified by several complementary factors. First, its small memory footprint and modular configuration allow inclusion only of the services required by the runtime, avoiding unnecessary overhead. This is consistent with Micro-Babel's objective of preserving predictable and bounded resource usage. Second, FreeRTOS provides a familiar and stable API for concurrency primitives such as tasks, queues, and timing services. These primitives serve as the foundation for Micro-Babel's event scheduling and timer management mechanisms, enabling them to operate with low jitter and without incurring the complexity of implementing custom scheduling logic at the hardware abstraction layer. Third, FreeRTOS has extensive support for integration with embedded networking stacks, including lightweight TCP/IP implementations and platform-specific drivers for Wi-Fi radios, which facilitates the communication management layer of Micro-Babel.

[25] https://www.freertos.org/

By building on FreeRTOS, Micro-Babel inherits these real-time and portability properties, while maintaining the core architectural commitments of the Babel ecosystem: explicit protocols, bounded event delivery, and interoperable decentralised execution across heterogeneous devices. Furthermore, to simplify the development we mapped each protocol running within a Micro-Babel process to a task of the FreeRTOS runtime, to which a blocking queue is associated. This allowed it to have a programming interface very similar to that of the variants in the Babel ecosystem, fully and naturally integrated into the core abstractions provided by FreeRTOS.

Event Representation and Dispatching

Events in Micro-Babel are represented using compact, statically defined data structures that include an event category, a subtype identifier, and an optional payload. The runtime preserves the four fundamental event classes used across the Babel ecosystem: message events, timer events, request events, and notification events.

Event dispatching is handled by a dedicated dispatcher component that routes events to interested protocols and runtime services based on explicit subscriptions, each of which is materialised as a (higher priority) FreeRTOS task. This mechanism enables one-to-many event dissemination while remaining predictable and bounded. Subscription tables are statically sized, and the maximum number of subscribers per event type is configurable at build time.

By centralising event dispatching, Micro-Babel decouples protocol logic from the mechanisms that generate events, allowing protocols to remain agnostic of platform-specific details.

Protocol Management

Protocol instances in Micro-Babel are managed by a protocol manager component responsible for registration, identification, and event/message routing. Protocols are registered during system initialisation and remain active for the duration of execution, reflecting the static nature of embedded deployments.

Each protocol maintains its own state and reacts independently to events delivered by the dispatcher. Protocol identifiers are embedded in message headers, enabling incoming messages to be demultiplexed and delivered to the appropriate handler without relying on heavyweight runtime reflection or dynamic dispatch. Furthermore, events whose interest has been registered by multiple protocols are managed by the Micro-Babel runtime to avoid duplicating them in memory. This was an essential approach to allow more complex applications developed on top of Micro-Babel to respect the low memory limits of integrated devices such as Raspberry Pico or ESP32.

This design supports the concurrent execution of multiple protocols on a single device, enabling complex decentralised behaviours to emerge from protocol composition even on constrained hardware.

Communication Management

Micro-Babel includes a communication management layer that abstracts the details of message transmission and reception across different networking technologies. Protocol

logic interacts with this layer through a uniform interface, while platform-specific adaptations handle the interaction with underlying networking stacks or radio drivers.

The communication manager is responsible for constructing message headers, handling protocol identifiers, and triggering message events upon reception. By isolating communication concerns within a dedicated component, Micro-Babel ensures that protocol implementations remain portable across platforms with different networking capabilities.

Network Management and Connectivity Awareness

A key aspect of Micro-Babel is the explicit treatment of network connectivity as a runtime concern. The network manager monitors link availability, connection status, and platform-specific networking events, and generates corresponding notification events when connectivity changes occur.

These events allow protocol logic to react dynamically to network disruptions, such as triggering neighbour re-discovery, adjusting dissemination strategies, or switching between communication modes. This design is particularly relevant for TaRDIS scenarios, where infrastructure degradation is expected rather than exceptional.

Platform Abstraction and Peripheral Integration

Platform-specific functionality, including timing services, concurrency primitives, and access to networking hardware, is isolated behind explicit abstraction layers. This separation allows the core runtime to remain platform-independent while supporting diverse embedded environments.

Micro-Babel also supports integration with device-level peripherals, such as sensors, buttons, and LEDs. These peripherals can act as event sources or sinks, enabling physical interactions to be incorporated into decentralised protocol execution. This capability is essential for demonstrators that combine sensing, actuation, and coordination under adverse conditions.

4.2.4 Relationship with Other WP6 Components

Micro-Babel complements the Babel-based contributions described in Section 4.1 by extending decentralised protocol execution to the lowest tiers of the system architecture. Embedded devices executing Micro-Babel can participate directly in decentralised communication and coordination, rather than relying on gateways or cloud services. Furthermore, this approach allows swarm systems that take advantage of these devices (relevant in contexts such as IoT, domotics, or factories) without requiring access to cloud services (or centralised infrastructures) or even to the Internet at all. This was a fundamental aspect leveraged in the plans developed in the Booster program to commercialize the Babel ecosystem through a spin-off.

This contribution also supported WP6's efforts on observability and telemetry aggregation by enabling embedded devices to generate, process, and disseminate runtime information locally. By explicitly exposing network and system events, Micro-Babel provides the foundation for integrating embedded-level observations into higher-level coordination and monitoring mechanisms developed elsewhere in WP6. We note,

however, that this has not been integrated into use cases throughout the duration of the activity of WP6.

4.2.5 Advances Beyond the State-of-the-Art

Existing embedded operating systems and IoT frameworks, such as TinyOS, Contiki-NG, RIOT, and FreeRTOS, provide event-driven execution models, lightweight networking stacks, and portability across constrained hardware platforms. TinyOS pioneered an event-centric programming model tailored to sensor networks, emphasising low power consumption and reactive execution.[26] Contiki and its successor Contiki-NG further explored lightweight operating system design with support for dynamic networking protocols and IP-based communication [27] [28]. RIOT similarly provides a modern OS abstraction for low-end IoT devices with real-time capabilities and modular networking support.[29] Finally, FreeRTOS, widely adopted in industrial embedded systems, offers deterministic task scheduling and a minimal kernel, making it suitable for deeply constrained environments[30].

While these platforms provide essential system-level services and communication primitives, they primarily focus on operating system concerns and low-level networking support. Decentralised coordination logic, protocol composition, and runtime interaction between multiple distributed protocols are typically implemented at the application level or delegated to external middleware. As a result, the explicit structuring, composition, and reuse of decentralised protocols are not treated as a first-class concern in these systems, which is the focus of Micro-Babel's design.

Complementarily, the distributed systems literature has produced a rich body of work on decentralised membership management, gossip-based dissemination, and epidemic communication protocols. Protocols such as HyParView[31] and Plumtree[32] demonstrate scalable and resilient approaches to peer-to-peer membership and broadcast dissemination. However, these protocols are typically specified and evaluated assuming relatively stable execution environments, homogeneous networking stacks, and the availability of general-purpose runtimes. Their direct applicability to embedded platforms

[26] Levis, P., Madden, S., Polastre, J., Szewczyk, R., Whitehouse, K., Woo, A., Gay, D., Hill, J., Welsh, M., Brewer, E., and Culler, D., “TinyOS: An Operating System for Sensor Networks,” in Ambient Intelligence, Springer, 2005

[27] Dunkels, A., Grönvall, B., and Voigt, T., “Contiki – A Lightweight and Flexible Operating System for Tiny Networked Sensors,” in Proceedings of the 29th IEEE International Conference on Local Computer Networks (LCN 2004), IEEE, 2004

[28] Oikonomou, G., Duquennoy, S., Elsts, A., Eriksson, J., Tanaka, Y., and Tsiftes, N., “The Contiki-NG Open Source Operating System for Next Generation IoT Devices,” SoftwareX, vol. 18, 2022

[29] Baccelli, E., Gündoğan, C., Hahm, O., Kietzmann, P., Petersen, H., Schleiser, K., Schmidt, T. C., and Wählisch, M., “RIOT: An Open Source Operating System for Low-End Embedded Devices in the IoT,” IEEE Internet of Things Journal, vol. 5, no. 6, 2018

[30] Guan, F., Peng, L., Perneel, L., and Timmerman, M., “Open Source FreeRTOS as a Case Study in Real-Time Operating System Evolution,” Journal of Systems and Software, vol. 118, 2016

[31] Leitão, J., Pereira, J., and Rodrigues, L., “HyParView: A Membership Protocol for Reliable Gossip-Based Broadcast,” in Proceedings of the 37th IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2007), IEEE, 2007

[32] Leitão, J., Pereira, J., and Rodrigues, L., “Epidemic Broadcast Trees,” in Proceedings of the 26th IEEE International Symposium on Reliable Distributed Systems (SRDS 2007), IEEE, 2007

operating under severe resource constraints, intermittent connectivity, and heterogeneous network interfaces remains limited.

Micro-Babel advances the state of the art by introducing a protocol-centric runtime model explicitly tailored to embedded devices, enabling decentralised protocol execution under strict resource constraints. Rather than embedding coordination logic directly into application code or relying on centralised middleware components, Micro-Babel makes decentralised protocols explicit, modular, and reusable units of behaviour, while preserving the event-driven execution semantics that underpin the Babel ecosystem.

By preserving decentralised execution semantics across heterogeneous platforms, Micro-Babel enables end-to-end decentralisation spanning embedded sensors, mobile devices, and edge systems. In contrast with existing approaches that treat embedded devices primarily as peripheral endpoints or data sources, Micro-Babel elevates them to first-class participants in decentralised coordination, thereby addressing a critical gap between embedded system platforms and decentralised systems middleware.

4.2.6 Experimental Validation and Evaluation

The validation of Micro-Babel primarily focuses on demonstrating its feasibility, robustness, and integration with other WP6 components. The runtime has been implemented and deployed on multiple embedded platforms representative of TaRDIS use cases, including Raspberry Pi Pico and ESP32-based devices.

Experimental validation confirms that Micro-Babel can support the concurrent execution of multiple protocols, handle dynamic network changes, and operate within the memory and processing constraints of target platforms. Integration experiments demonstrate interoperability with Babel-based components running on more powerful devices, enabling heterogeneous decentralised deployments.

A detailed quantitative evaluation of performance and scalability could not be conducted at the time of writing, primarily because resorting to emulation platforms in the context of Micro Babel is not a suitable solution. Moreover, given the runtime's exploratory nature and the diversity of target platforms, we are still evolving this runtime and improving its integration with the Babel ecosystem (this was in part motivated by work conducted within the context of the Booster program). Nevertheless, the results obtained from the application case study developed in the context of WP6 provide strong evidence that decentralised protocol execution on embedded devices is feasible and practical, and that Micro-Babel constitutes a solid foundation for further development and exploitation beyond the TaRDIS project.

4.2.7 Limitations and Lessons Learned

The development of Micro-Babel exposed a number of limitations intrinsic to executing decentralised protocol abstractions on highly constrained embedded platforms, as well as limitations stemming from the runtime's current design choices and implementation scope. These limitations should be understood in the context of the exploratory and foundational nature of this contribution, whose primary objective was to assess feasibility and identify a viable design space rather than deliver fully mature embedded middleware.

A first limitation concerns the degree of static configuration required by the current Micro-Babel implementation. In order to ensure predictable resource usage and

compatibility with microcontroller-based platforms, several aspects of the runtime - including protocol registration, event queue sizes, and subscription tables - are fixed at build time. While this approach is well aligned with embedded system constraints, it restricts dynamic extensibility and limits the ability to adapt protocol compositions at runtime. This limitation contrasts with the more flexible lifecycle management supported by the Babel runtime on general-purpose platforms, highlighting a fundamental trade-off between adaptability and predictability in constrained environments.

Another important limitation arises from the heterogeneity of networking stacks and capabilities across supported platforms. While Micro-Babel deliberately abstracts communication details behind a uniform interface, differences in available transports, buffer management strategies, and radio characteristics still influence the behaviour and performance of decentralised protocols. As a result, protocol implementations that behave similarly on general-purpose platforms may exhibit different performance profiles when executed on embedded devices. This observation justifies the need for platform-aware tuning and highlights that full transparency across heterogeneous devices remains difficult to achieve in practice. In the context of TaRDIS's activities, we did not focus on this aspect.
From a broader perspective, the development of Micro-Babel also revealed the challenges of maintaining architectural consistency across multiple runtime variants targeting very different execution environments. While conceptual alignment with the Babel ecosystem was preserved, several abstractions had to be simplified or reinterpreted to fit the constraints of embedded platforms.

Despite these limitations, the lessons learned from Micro-Babel are largely positive. The work demonstrates that decentralised protocol execution is feasible on highly constrained devices without resorting to centralised coordination or application-specific shortcuts. It also shows that a protocol-centric design approach can be meaningfully extended beyond servers and mobile devices, enabling embedded nodes to participate as first-class components in decentralised systems. These insights directly inform the ongoing evolution of the Babel ecosystem and provide a solid foundation for future work beyond the scope of TaRDIS.

4.3 Decentralised Membership for Dynamic and Intermittent Networks

The contributions reported in this section address a core enabling capability for decentralised swarm systems: the maintenance and exploitation of membership-related information under conditions of extreme network dynamics and constrained communication opportunities. While the runtime and communication abstractions described in Sections 4.1 and 4.2 provide the necessary execution substrate for decentralised protocols across heterogeneous platforms, effective coordination in swarm environments also requires mechanisms that allow nodes to reason about the presence, reachability, and evolution of other participants over time.

Within WP6, this contribution is primarily associated with Task 6.1, complementing the communication and coordination mechanisms developed elsewhere in the work package by focusing explicitly on decentralised dissemination and membership-related reasoning under adverse and highly dynamic network conditions. It is also related with Task 6.3, as the ability to disseminate information reliably and predictably under intermittent

connectivity directly influences higher-level observability, aggregation, and monitoring mechanisms.

The design and evaluation of this contribution are strongly motivated by the satellite swarm use case explored in TaRDIS in collaboration with GMV. In this context, communication opportunities between nodes are governed by orbital dynamics and line-of-sight constraints, resulting in predictable yet highly transient connectivity patterns. These characteristics expose fundamental limitations in existing decentralised dissemination and membership protocols, which are typically designed for environments where connectivity persists long enough for overlay maintenance mechanisms to stabilise. The work reported here revisits decentralised dissemination and membership-related mechanisms under these constraints, treating dynamic visibility not as an exceptional condition but as a fundamental aspect of the system model.

4.3.1 Context and Motivation

Satellite swarms operate in environments where network connectivity is inherently intermittent, structured, and externally determined by physical dynamics. Unlike terrestrial networks, where connectivity fluctuations are often modelled as failures or stochastic disturbances, satellite visibility patterns are predictable but short-lived, and may change rapidly as satellites move along their orbits. As a result, decentralised protocols deployed in such environments must operate effectively despite frequent topology changes and limited communication windows.

Classical membership and dissemination protocols typically assume that nodes can exchange messages over sufficiently long periods to establish and maintain overlay structures. In satellite swarms, these assumptions do not hold: nodes may be mutually visible only for short intervals, and global connectivity may never be available simultaneously. Under these conditions, protocols that rely on stable neighbourhoods or long-lived connections risk failing to disseminate information effectively or converging too slowly to be useful.
The motivation for this contribution is therefore to explore decentralised dissemination and membership-related mechanisms that can operate under visibility-constrained conditions, allowing nodes to propagate state information opportunistically as connectivity becomes available. Rather than attempting to maintain a globally consistent membership view, the focus is on enabling effective information dissemination and state convergence over time, despite the absence of stable end-to-end paths.

4.3.2 Objective and Design Goals

The primary objective of this contribution is to design and evaluate decentralised dissemination mechanisms to support swarm coordination under highly dynamic and intermittently connected environments. The work aims to characterise how information propagates in such environments and to identify protocol parameters and design choices that influence convergence, reliability, and efficiency.

Key design goals include minimising reliance on long-lived connections, tolerating frequent topology changes without global coordination, and enabling dissemination progress even when connectivity is fragmented. The design explicitly avoids assumptions

of continuous connectivity or centralised control, aligning with the operational realities of satellite swarms.

Rather than targeting strong guarantees such as complete and instantaneous dissemination, the contribution focuses on probabilistic and eventual properties, analysing how dissemination quality evolves over time and how it is affected by swarm size, connectivity patterns, and protocol parameters.

4.3.3 Technical Description

The technical solution combines a lightweight membership service with an epidemic dissemination mechanism, both designed to operate under intermittent connectivity. Each node maintains a local view of other nodes it has recently interacted with, based on opportunistic exchanges during visibility windows. This local view is not intended to represent the full swarm membership, but rather to support neighbour selection and dissemination decisions.

Dissemination follows a gossip-based approach, where messages are propagated opportunistically whenever nodes become mutually visible. Each node forwards received messages to a subset of peers, subject to configurable fanout and time-to-live parameters. This design allows information to gradually spread across the swarm as visibility windows overlap, even if no stable end-to-end paths exist at any given time.

Membership-related information is embedded within dissemination exchanges, allowing nodes to update their local views as they interact with different peers over time. The protocol does not attempt to enforce a consistent global membership view; instead, it leverages the cumulative effect of repeated local interactions to enable information propagation and eventual convergence.

Crucially, the design treats visibility constraints as a first-class concern. Dissemination decisions are driven by actual communication opportunities rather than by assumed overlay structures, and the protocol tolerates message loss, reordering, and duplication as natural consequences of intermittent connectivity.

4.3.4 Relationship with Other WP6 Components

This contribution builds directly on the decentralised communication abstractions provided by the Babel ecosystem, using them to structure message exchange and protocol execution. It complements the membership mechanisms described in Section 4.1 by focusing on dissemination behaviour under extreme dynamics rather than on maintaining stable neighbourhoods.

The dissemination properties analysed here also inform the design of higher-level components developed under WP6, including replicated data management and decentralised telemetry aggregation. In particular, understanding how information propagates under visibility constraints is essential for interpreting monitoring data and for designing aggregation mechanisms that remain effective despite partial and delayed dissemination.

4.3.5 Advances Beyond the State-of-the-Art

Existing epidemic dissemination and membership protocols are typically evaluated under assumptions of relatively stable connectivity or random failure models. While such approaches perform well in peer-to-peer and data-centre environments, they do not directly address scenarios in which connectivity is both intermittent and subject to external constraints, such as orbital dynamics.

The contribution reported here advances the state of the art by explicitly evaluating decentralised dissemination mechanisms under a connectivity model representative of satellite swarms. Rather than treating disconnections as failures, the evaluation incorporates visibility windows as a fundamental aspect of the system model, enabling a more realistic assessment of protocol behaviour.

Moreover, the work provides empirical insights into how dissemination quality evolves as swarm size increases and connectivity becomes more constrained. By analysing metrics such as convergence, hop count, propagation delay, and delivery ratio under different configurations, the contribution sheds light on trade-offs that are not captured by traditional evaluation models.

4.3.6 Experimental Validation and Evaluation

The experimental evaluation was designed to assess the behaviour of the dissemination and membership-related mechanisms under conditions representative of satellite swarms. Experiments were conducted in a controlled cloud-based environment using containerised deployments, allowing the execution of large-scale scenarios with reproducible configurations.

Each experimental scenario models a satellite swarm with a fixed number of nodes, where connectivity between nodes is determined by precomputed visibility patterns derived from orbital dynamics. Nodes are deployed as containers and execute identical protocol logic, interacting only when visibility permits communication. Scenarios were evaluated for different swarm sizes, notably configurations corresponding to approximately 135, 150, and 170 nodes, reflecting increasing system scale.

The evaluation focuses on a set of metrics that capture dissemination behaviour and convergence properties. State convergence measures how quickly nodes obtain a consistent estimate of the disseminated state over time. Node coverage captures the fraction of nodes that successfully receive a given message as dissemination progresses. Hop count measures the number of forwarding steps required for messages to reach different parts of the swarm, providing insight into dissemination efficiency. Propagation delay quantifies the time it takes for messages to propagate across visibility-constrained paths, while delivery ratio measures the proportion of messages that are eventually delivered to their intended recipients.

The results show that the progress of dissemination is strongly influenced by swarm size and connectivity patterns. For smaller configurations, state convergence and node coverage increase steadily over time, indicating effective exploitation of overlapping visibility windows. As swarm size increases, convergence slows and coverage plateaus at lower values, reflecting a reduced probability of overlapping communication opportunities.

Hop count measurements reveal that dissemination paths remain relatively short even as the swarm grows, although variability increases in larger configurations. Propagation delay exhibits significant variance, driven by the timing and alignment of visibility windows rather than by protocol inefficiencies alone. Delivery ratio decreases as swarm size increases, highlighting the inherent limitations imposed by constrained connectivity.

Overall, the evaluation demonstrates that decentralised dissemination under visibility-constrained conditions is feasible, but subject to fundamental trade-offs between scale, timeliness, and reliability. These results provide valuable guidance for the design of higher-level coordination and data management mechanisms in satellite swarm environments.

Results and Discussion

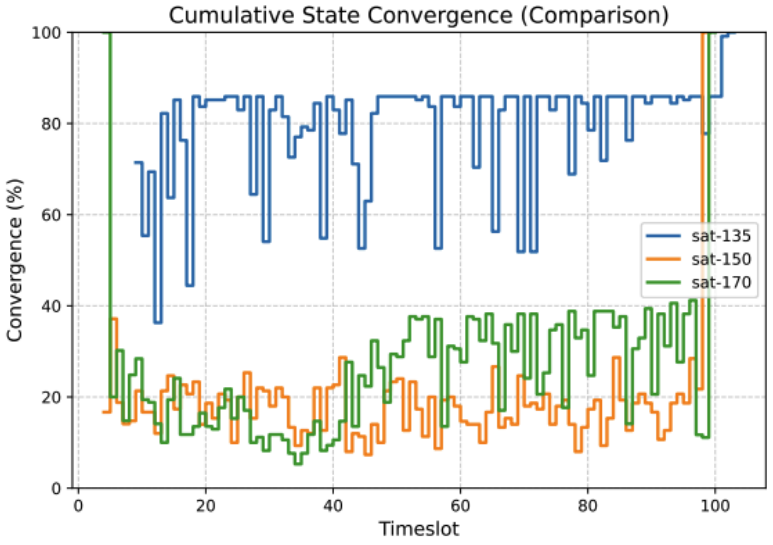

Figure 1: *Cumulative State Convergence.*

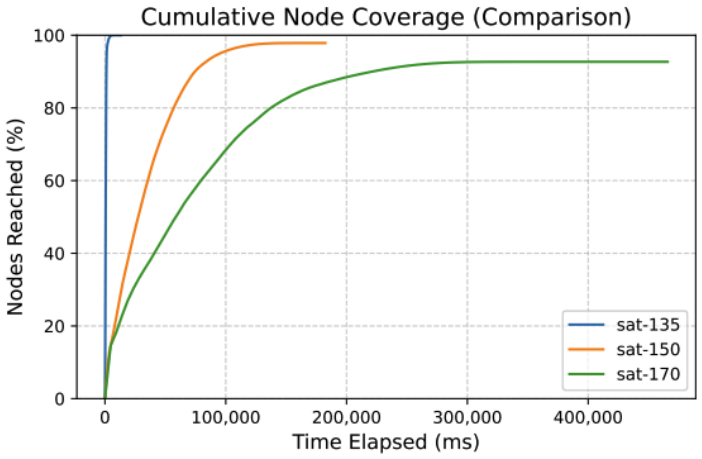

Figure 2: *Cumulative Node Coverage.*

The evolution of membership coverage over time is illustrated in Figures 1 and 2. These figures show that, despite intermittent connectivity, membership information propagates progressively through the swarm as visibility windows overlap. Coverage increases in a stepwise fashion, reflecting the episodic nature of communication opportunities. While full

coverage is not always achieved, especially in larger swarms, local views rapidly reach a level sufficient to support coordination tasks.

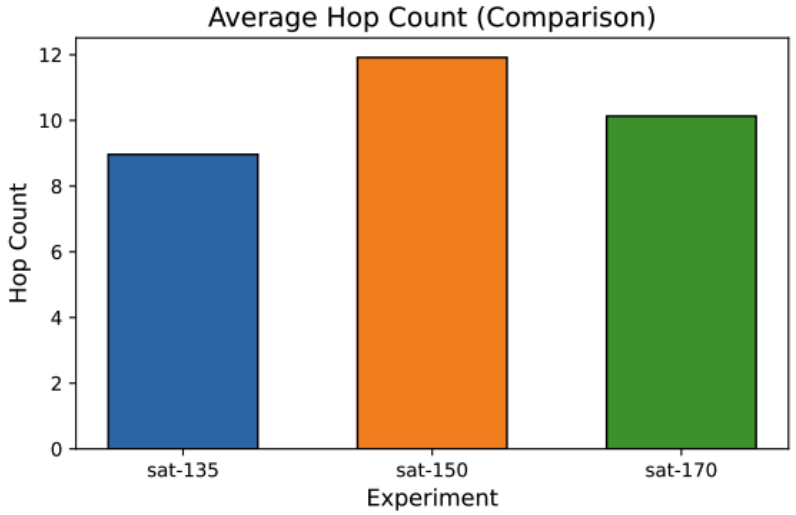

Figure 3: *Average Hop Count.*

The impact of swarm size on the number of communication steps required to propagate information throughout the entire swarm is shown in Figure 3. As swarm size increases, the number of communication hops increases, leading to additional delays in information dissemination. Surprisingly, when the swarm size increases from 150 to 170 nodes, the number of communication hops required to propagate information decreases, which can be explained by the emergence of additional communication opportunities among the devices in the swarm.

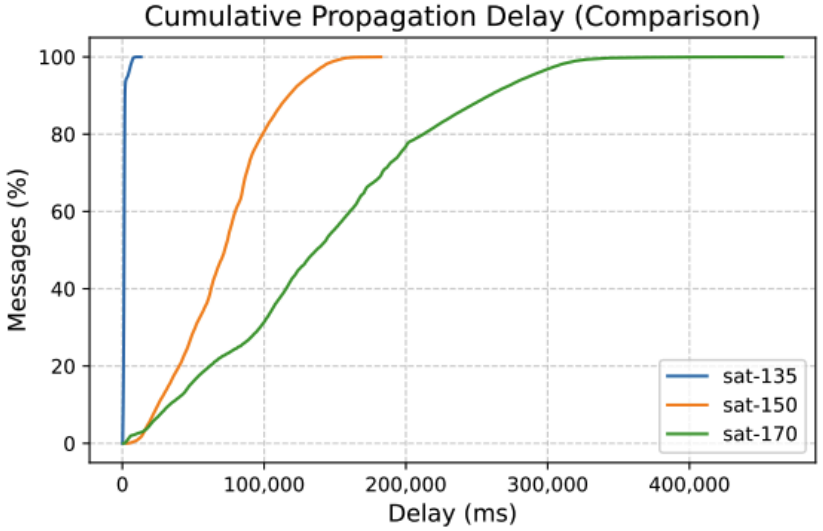

Figure 4: *Cumulative Propagation Delay.*

Complementary to the results discussed above, Figure 4 presents the cumulative distribution function for the delivery of messages disseminated across three swarms of different sizes. This shows that the epidemic dissemination strategy, on top of our proposed reactive membership, can deliver information across the entire swarm most of the time, although the time required for nodes to propagate information increases noticeably with the total swarm size.

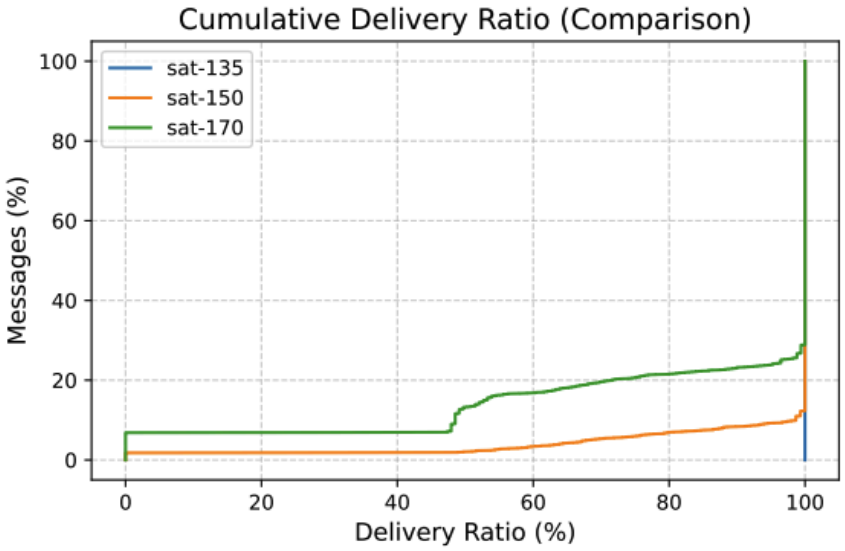

Figure 5: *Cumulative Delivery Ratio.*

Figure 5 reports on the cumulative distribution function of the fraction of messages that are able to be delivered to at least a given fraction of active nodes in the swarm, which we encode as the delivery ratio. The results show that a small fraction of messages reaches about half of the nodes in swarms of 150 or 170 nodes. Then, a few messages up to 20% of total disseminated messages reach less than 100 nodes for these two swarm sizes, with a clear penalty for the swarm composed of only 150 nodes. For the smaller swarm considered (135 nodes), all messages were successfully delivered to all nodes.

Summary of Evaluation Results

Overall, the experimental results demonstrate that decentralised membership maintenance is feasible under visibility-constrained connectivity. While the protocol does not provide strong consistency or complete views, it enables nodes to maintain evolving, sufficiently rich membership information to support coordination in dynamic swarm environments by leveraging gossip communication strategies. The evaluation also highlights inherent trade-offs and scalability limits, providing valuable guidance for designing higher-level coordination mechanisms in satellite swarms.

4.3.7 Limitations and Lessons Learned

The results reported in this section highlight several limitations inherent to decentralised dissemination in highly dynamic and intermittent networks. First, dissemination quality degrades as swarm size increases, reflecting the reduced overlap of visibility windows and the absence of stable end-to-end paths. This limitation is fundamental to the environment and cannot be fully addressed by protocol design alone.

Second, the reliance on opportunistic interactions implies that dissemination latency and coverage are inherently variable. While the protocol ensures progress over time, it cannot guarantee timely or complete dissemination in all scenarios. Applications built on top of these mechanisms must therefore tolerate partial and delayed information.

Finally, the evaluation underscores the importance of realistic connectivity models. Protocols that perform well under idealised assumptions may behave very differently

when deployed in structured, intermittent connectivity. The lessons learned from this contribution informed subsequent WP6 work on data replication and telemetry aggregation, reinforcing the need to design decentralised mechanisms that are robust to extreme dynamics.

4.4 Advanced Replicated Data Management at the Edge

The contributions reported in this section address one of the most challenging aspects of decentralised edge systems: the management of shared and replicated data under conditions of scale, heterogeneity, and adversarial behaviour. While decentralised communication, membership, and execution support are essential to enable cooperation among distributed entities, meaningful application-level functionality ultimately depends on the ability to store, update, and observe shared state in a way that remains robust to failures and attacks.
Within WP6, this contribution is primarily associated with Task 6.2, which focuses on data management mechanisms for decentralised environments. It builds directly on the communication and membership abstractions described in Sections 4.1 and 4.3 and interacts closely with the monitoring and observability mechanisms developed under Task 6.3. The work reported here is particularly relevant for edge-centric scenarios in which data is produced, consumed, and replicated across a large number of geographically distributed nodes that cannot rely on strong trust assumptions or centralised control.

A distinguishing feature of this contribution is its explicit consideration of the Byzantine fault model in large-scale edge deployments. Unlike cloud data centres, edge environments often lack strong physical and administrative security perimeters, making it realistic to assume that some nodes may behave arbitrarily, either due to compromise or misconfiguration. The mechanisms presented in this section therefore revisit classical replication and consistency techniques, adapting them to settings where both failures and malicious behaviour must be tolerated, while still preserving scalability and acceptable performance.

4.4.1 Context and Motivation

Edge computing environments introduce a fundamental tension between proximity and trust. By bringing data and computation closer to end users, edge deployments reduce latency and enable new classes of applications. At the same time, edge nodes are typically more exposed, heterogeneous, and operationally diverse than machines deployed in controlled data centre environments. As a result, assumptions that underpin many existing distributed storage systems—such as uniform trust, stable connectivity, or reliable membership—no longer hold.

Traditional cloud storage systems rely on replication to mask failures and improve availability, often assuming crash-fault models and controlled administrative domains. While such assumptions are reasonable in data centres, they are increasingly problematic at the edge, where nodes may be intermittently connected, resource-constrained, or compromised. Simply extending cloud-based replication

techniques to the edge risks either excessive overhead or insufficient protection against malicious behaviour.

The motivation for this contribution arises from the need to reconcile three competing requirements. First, replicated data at the edge must remain available and responsive despite failures and churn. Second, the system must tolerate Byzantine faults without relying on heavyweight consensus protocols that scale poorly or impose prohibitive latency. Third, replication mechanisms must accommodate partial replication and heterogeneous node capabilities, reflecting the uneven distribution of resources typical of edge environments.

4.4.2 Objective and Design Goals

The primary objective of this contribution is to design and evaluate a replicated data management mechanism suitable for large-scale edge deployments operating under the Byzantine fault model. Rather than targeting strong consistency guarantees that are difficult to maintain at scale, the work focuses on consistency models that provide meaningful application-level semantics while preserving decentralisation and performance.

The design is guided by several key goals. Replication should be decentralised and scalable, avoiding global coordination or central authorities. The system should support partial replication, allowing data to be stored and processed only on subsets of nodes relevant to a given application or workload. Consistency guarantees should be expressive enough to support non-trivial application logic, while remaining compatible with asynchronous execution and intermittent connectivity. Finally, the mechanisms should be resilient to Byzantine behaviour, ensuring that malicious nodes cannot easily violate consistency or corrupt replicated state.

4.4.3 Technical Description

The replicated data management mechanisms developed in this contribution are designed to operate in large-scale edge environments characterised by decentralisation, partial trust, and heterogeneous node capabilities. The technical solution is structured around three core design principles: partial replication, causality-aware consistency, and Byzantine-resilient update dissemination. Together, these principles define a replication model that avoids global coordination while providing meaningful guarantees in adversarial settings.

Architectural Overview

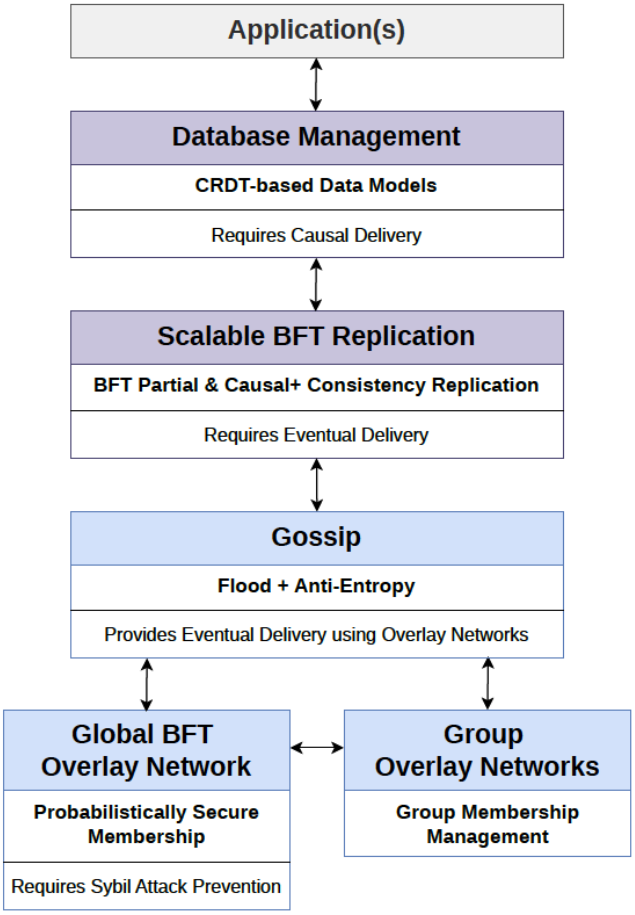

Figure 6: *Data Management Architecture.*

The system adopts a layered architecture (detailed in Figure 6) in which replication logic is decoupled from communication and membership management. At the base, the system relies on decentralised communication primitives and membership services to establish and maintain knowledge about participating nodes. On top of this substrate, the replication layer is responsible for managing data placement, propagating updates, enforcing consistency constraints, and mitigating Byzantine behaviour.

Each node hosts a replication component responsible for the data items assigned to it. Nodes may act as full replicas, partial replicas, or observers, depending on their role and resource constraints. This flexibility reflects the heterogeneity inherent to edge environments and avoids imposing uniform storage or computation requirements across all participants.

Partial Replication Model

Unlike traditional fully replicated systems, where every replica stores all data items, the proposed solution supports partial replication as a first-class concept. Each data item is associated with a replica set, defined as the subset of nodes responsible for storing and processing updates to that item. Replica sets may overlap and evolve over time, but their size is bounded to limit communication and storage overhead.

Partial replication significantly reduces resource consumption and improves scalability, but it introduces challenges related to consistency and fault tolerance. In particular, updates must be disseminated only to relevant replicas, and consistency guarantees must be maintained without assuming global visibility. The system addresses these challenges by explicitly tracking causal dependencies and by embedding fault-tolerance mechanisms into the dissemination process.

Causality-Aware Consistency

To avoid the overhead and coordination requirements of strong consistency models, the system adopts a causality-based consistency model. Each update is associated with metadata that captures its causal dependencies, allowing replicas to determine the correct application order without relying on total ordering or central coordination.

Causal metadata is maintained using compact dependency tracking structures that summarise the set of updates known to a replica. When an update is propagated, its associated metadata allows receiving replicas to determine whether all required dependencies are satisfied. If dependencies are missing, updates may be buffered until they can be safely applied.

This approach ensures that causally related updates are observed in a consistent order across replicas, while allowing concurrent updates to be applied independently. The resulting consistency model is expressive enough to support a wide range of application semantics, yet lightweight enough to scale to large, decentralised deployments.

Update Dissemination and Replica Coordination

Update dissemination follows a decentralised, gossip-inspired approach that leverages membership information to scope communication. When a replica generates or receives an update, it propagates the update to other replicas in the corresponding replica set, subject to resource and connectivity constraints.

Dissemination is opportunistic and asynchronous. Replicas do not assume reliable or timely delivery, and updates may arrive out of order or be duplicated. The combination of causal metadata and idempotent update processing ensures correctness under these conditions.

Replica coordination avoids explicit consensus protocols. Instead, correctness relies on the eventual dissemination of updates and on the enforcement of causality constraints at each replica. This design choice is essential for scalability and is particularly well suited to edge environments with intermittent connectivity.

Byzantine Fault Model and Threat Assumptions

The system explicitly considers the Byzantine fault model, in which nodes may behave arbitrarily, including sending malformed or conflicting updates, omitting updates, or attempting to subvert consistency guarantees. Rather than assuming trusted replicas, the design incorporates defensive mechanisms that limit the influence of faulty nodes.

Each update is cryptographically authenticated, allowing replicas to verify the origin and integrity of the information they receive. Replicas validate updates against consistency

rules and causal dependencies before applying them, ensuring that invalid or contradictory updates are detected and discarded.

Byzantine replicas may attempt to inject conflicting updates or withhold information, but their impact is confined by the structure of replica sets and by the decentralised nature of dissemination. Correct replicas can continue to make progress as long as a sufficient fraction of the replica set behaves correctly, without requiring global agreement or leader election.

Data Types and Application Semantics

In addition to the replication and dissemination mechanisms, the system provides abstractions for constructing replicated data types whose semantics remain well-defined under concurrent updates and partial replication. These abstractions allow application developers to reason about shared state without directly managing causal metadata or fault tolerance mechanisms.

Data types are designed to be monotonic or to resolve conflicts deterministically, ensuring convergence despite concurrency and network delays. This design aligns with the broader trend towards conflict-tolerant data abstractions in decentralised systems, while extending them to operate under Byzantine assumptions and partial replication.

Integration with Observability and Management

The replication layer exposes internal metrics and state transitions that can be consumed by observability and monitoring mechanisms developed elsewhere in WP6. These include metrics related to update propagation delays, metadata growth, replica availability, and detected faulty behaviour.

This integration enables higher-level management strategies, such as adaptive replica placement or dynamic tuning of dissemination parameters, although such strategies are beyond the scope of the current contribution.

4.4.4 Relationship with Other WP6 Components

This contribution builds directly on the communication and coordination mechanisms provided by the Babel ecosystem, using them to structure message exchange and protocol execution. It relies on decentralised membership services to identify replica sets and to scope dissemination activities, and it interacts with telemetry and monitoring components to expose runtime information about replication behaviour and fault conditions.

Conversely, the replicated data management mechanisms developed here inform the requirements placed on other WP6 components. In particular, they highlight the need for flexible membership abstractions, efficient dissemination primitives, and observability mechanisms that capture consistency and fault-related metrics. The contribution, therefore, plays a central role in integrating communication, coordination, and data management within WP6.

4.4.5 Advances Beyond the State-of-the-Art

Replicated data management has been extensively studied in distributed systems, with a large body of work focusing on scalable replication under crash-fault assumptions in

data-centre environments. Systems such as Dynamo introduced a decentralised replication architecture prioritising availability and partition tolerance, relying on techniques such as sloppy quorums and version vectors to reconcile divergent updates[33]. While highly influential, these designs deliberately avoid strong consistency and assume benign failures, making them unsuitable for environments where adversarial behaviour must be considered.

Subsequent work explored stronger yet scalable consistency models, most notably causal consistency. Systems such as COPS demonstrated that causal consistency can be achieved at scale with low latency by explicitly tracking dependencies between updates and avoiding global coordination[34]. Follow-up systems, including Eiger and Orbe, further optimised dependency tracking and propagation to reduce overhead and improve performance[35] [36]. Despite their scalability, these systems assume trusted replicas, stable membership, and controlled deployment environments, assumptions that do not hold in large-scale edge systems.

Conflict-Free Replicated Data Types (CRDTs) represent another influential line of work, enabling replicas to converge automatically under concurrent updates without coordination[37]. CRDTs provide elegant convergence guarantees and are well-suited to decentralised and partition-tolerant settings. However, most CRDT-based systems operate under crash-fault assumptions and do not address malicious behaviour. Moreover, many CRDT designs implicitly assume full replication or dissemination patterns that do not align well with partial replication and heterogeneous edge environments.

Byzantine fault-tolerant storage systems address adversarial behaviour by assuming that replicas may behave arbitrarily. Classical approaches rely on Byzantine consensus protocols, such as PBFT, to ensure safety and agreement[38]. While effective in small, well-provisioned deployments, these protocols impose substantial communication and computational overhead and scale poorly in asynchronous, large-scale environments. As a result, their applicability to edge deployments with intermittent connectivity and heterogeneous nodes is limited.

More recent research has explored combining weaker consistency models with Byzantine fault tolerance to reduce coordination costs while preserving security guarantees. These approaches typically relax consistency or constrain the fault model,

[33] DeCandia, G., Hastorun, D., Jampani, M., et al., “Dynamo: Amazon’s Highly Available Key-Value Store,” Proceedings of the 21st ACM Symposium on Operating Systems Principles (SOSP), 2007.
[34] Lloyd, W., Freedman, M. J., Kaminsky, M., and Andersen, D. G., “Don’t Settle for Eventual: Scalable Causal Consistency for Wide-Area Storage with COPS,” Proceedings of the 23rd ACM Symposium on Operating Systems Principles (SOSP), 2011
[35] Mahajan, P., Alvisi, L., and Dahlin, M., “Consistency, Availability, and Convergence,” University of Texas at Austin Technical Report, 2011
[36] Du, J., Iorgulescu, C., Roy, S., and Zwaenepoel, W., “Orbe: Scalable Causal Consistency Using Dependency Matrices and Physical Clocks,” Proceedings of the 4th ACM Symposium on Cloud Computing (SoCC), 2013
[37] Shapiro, M., Preguiça, N., Baquero, C., and Zawirski, M., “Conflict-Free Replicated Data Types,” Proceedings of the 13th International Symposium on Stabilization, Safety, and Security of Distributed Systems (SSS), 2011
[38] Castro, M. and Liskov, B., “Practical Byzantine Fault Tolerance,” Proceedings of the 3rd USENIX Symposium on Operating Systems Design and Implementation (OSDI), 1999

but they often assume static replica groups, homogeneous nodes, or limited system scale. Consequently, they do not fully address the challenges posed by decentralised edge systems that combine partial replication, dynamic membership, and adversarial behaviour.[39]

The contribution reported in this section advances the state of the art by demonstrating that causality-aware consistency, partial replication, and Byzantine fault tolerance can be combined in a decentralised and scalable manner without relying on global consensus. By adopting causal consistency rather than strong consistency, the system avoids the synchronisation overhead of Byzantine agreement. By supporting partial replication, it accommodates heterogeneous edge environments and reduces resource consumption. By embedding Byzantine resilience directly into update validation and dissemination mechanisms, this approach constrains the influence of malicious nodes without requiring global coordination.

Crucially, this work integrates replicated data management with decentralised membership and communication services, rather than treating storage as an isolated subsystem. This holistic approach reflects the operational realities of edge and swarm systems and extends existing replication techniques beyond data-centre settings. In doing so, it provides a practical and previously underexplored design point for secure and scalable data management in decentralised edge deployments, directly supporting the TaRDIS objectives and use cases.

4.4.6 Experimental Validation and Evaluation

The experimental evaluation of the replicated data management mechanisms was designed to assess their scalability, performance, and resilience under both benign and adversarial conditions, with particular emphasis on large-scale edge deployments. The evaluation focuses on understanding the cost of combining partial replication, causal consistency, and Byzantine fault tolerance, and on characterising the trade-offs that arise from these design choices.

Experimental Setup

The experiments were conducted using an emulated distributed environment in which each logical node executes an instance of the replication protocol. Nodes were deployed as independent processes within a controlled infrastructure, allowing the system to scale to several hundred replicas while preserving reproducibility and precise control over fault injection.

The deployment models a decentralised edge environment in which replicas communicate over an asynchronous network. Message delays and reordering are permitted, but messages are not lost unless explicitly dropped as part of a fault scenario. Replica sets are statically defined for the duration of each experiment, enabling controlled evaluation of partial replication configurations.

Each node stores a subset of the global data set, according to the partial replication model described in Section 4.4.3. Updates are generated at configurable rates and

[39] Albert van der Linde, João Leitão, and Nuno Preguiça. 2020. Practical client-side replication: weak consistency semantics for insecure settings. Proc. VLDB Endow. 13, 12 (August 2020), 2590–2605. https://doi.org/10.14778/3407790.3407847

propagated to the relevant replica sets via the system's decentralised dissemination mechanisms. Causal metadata is attached to each update and used to enforce consistency constraints at replicas.

To evaluate Byzantine resilience, a configurable fraction of replicas is designated as faulty. Faulty replicas may exhibit arbitrary behaviour, including sending malformed updates, omitting updates, or attempting to violate causal dependencies. Correct replicas validate received updates and discard invalid or inconsistent information.

Metrics and Evaluation Criteria

The evaluation focuses on several key metrics that capture both performance and correctness aspects.

The first metric is operation latency, measured as the time between the generation of an update and its successful application at a replica. This metric reflects the system's responsiveness under varying loads and fault conditions.

The second metric is throughput, measured as the number of updates successfully processed per unit of time. Throughput captures the system’s ability to sustain concurrent updates as the number of replicas increases.

The third metric is metadata overhead, measured by the size of causal metadata associated with updates and the memory footprint required to maintain dependency-tracking structures. This metric is particularly important in edge environments, where resources are constrained.

The fourth metric is fault impact, measured by observing how Byzantine replicas affect the visibility and application of updates at correct replicas. This includes the rate of rejected updates and the ability of correct replicas to continue making progress despite malicious behaviour.

Evaluation Scenarios

The experimental campaign explores multiple scenarios by varying system size, replication degree, update rates, and fault configurations. Experiments range from small deployments with a few tens of replicas to larger configurations involving several hundred replicas.

Partial replication is evaluated by varying replica set sizes, enabling analysis of how the degree of replication influences performance and metadata growth. Fault scenarios include both crash faults and Byzantine faults, enabling comparison between benign and adversarial conditions.

Baseline comparisons are performed against a replicated storage configuration that provides causal consistency under crash-fault assumptions but does not include Byzantine resilience mechanisms. This comparison isolates the cost of Byzantine fault tolerance and highlights the overhead introduced by additional validation and verification steps.

Results and Discussion

We now present and briefly discuss the main experimental results that we have obtained for this contribution.

Metadata Volume and Partial Replication Effects

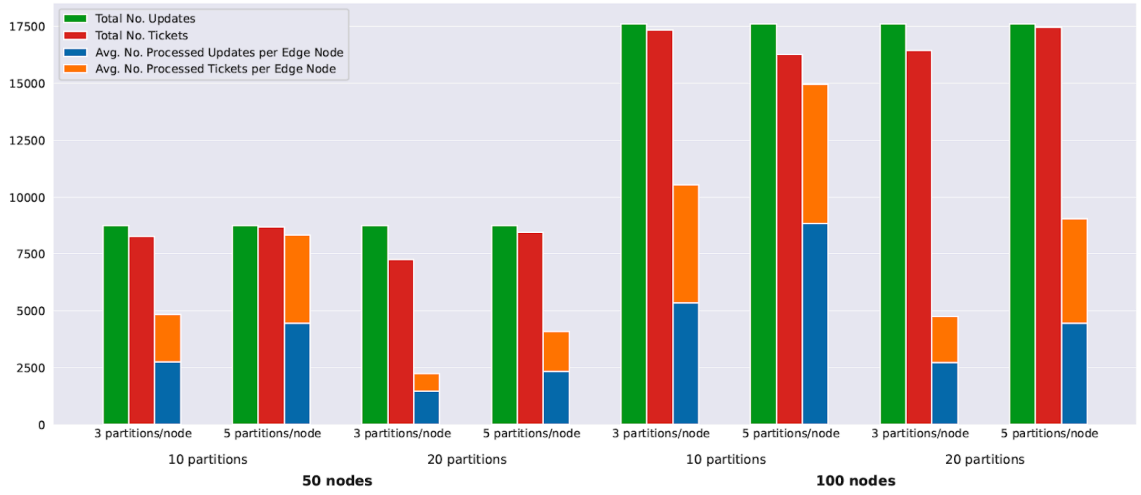

Figure 7: *Protocol Metadata Volume.*

Figure 7 reports the volume of protocol metadata processed by edge nodes under different system configurations, distinguishing between application updates and validation tickets. The results are presented for varying numbers of nodes, partitions, and replication degrees, allowing the impact of partial replication to be examined explicitly.

The stacked bars show that, for most configurations, each node processes only a fraction of the total updates generated in the system, confirming that partial replication effectively limits the dissemination scope of updates. Even when accounting for ticket messages required to validate inter-partition dependencies, the total number of messages processed per node remains significantly lower than the global update volume, demonstrating that partiality is preserved despite the additional security mechanisms.

However, the figure also shows that when nodes replicate a larger fraction of the total partitions, the number of tickets approaches the number of updates, reducing the benefits of partial replication. This highlights a key trade-off: partial replication is most effective when nodes are responsible for relatively small portions of the global data set, a condition that aligns well with heterogeneous edge environments.

Dependency Overhead and Concurrency Effects

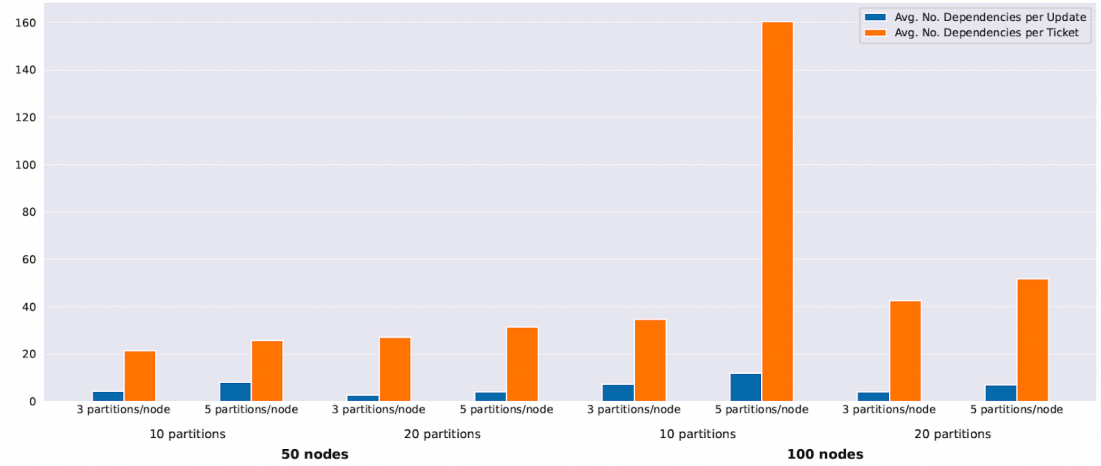

Figure 8: *Protocol Dependency Volume.*

Figure 8 characterizes the dependency overhead associated with causal tracking, reporting the average number of dependencies attached to updates and tickets across different configurations. Dependencies correspond to cryptographic identifiers of causally preceding operations and directly influence metadata size and processing cost.

The results show that updates typically have a limited number of dependencies, reflecting that newly generated updates usually extend the heads of local dependency graphs. As a consequence, the number of dependencies per update is bounded by the system's concurrency level.

In contrast, tickets exhibit a higher and more variable dependency count, particularly as the number of nodes and replicated partitions increases. This behaviour is expected, as tickets are generated to capture inter-partition dependencies and must aggregate causal information across multiple topics. Under heavier load, ticket generation becomes a bottleneck, leading to increased concurrency and, consequently, larger dependency sets.

These results confirm that dependency growth is primarily driven by concurrency and the degree of replication, rather than by system size alone, and that ticket processing is a critical scalability factor in Byzantine-resilient causal replication.

Throughput–Latency Trade-offs under Load

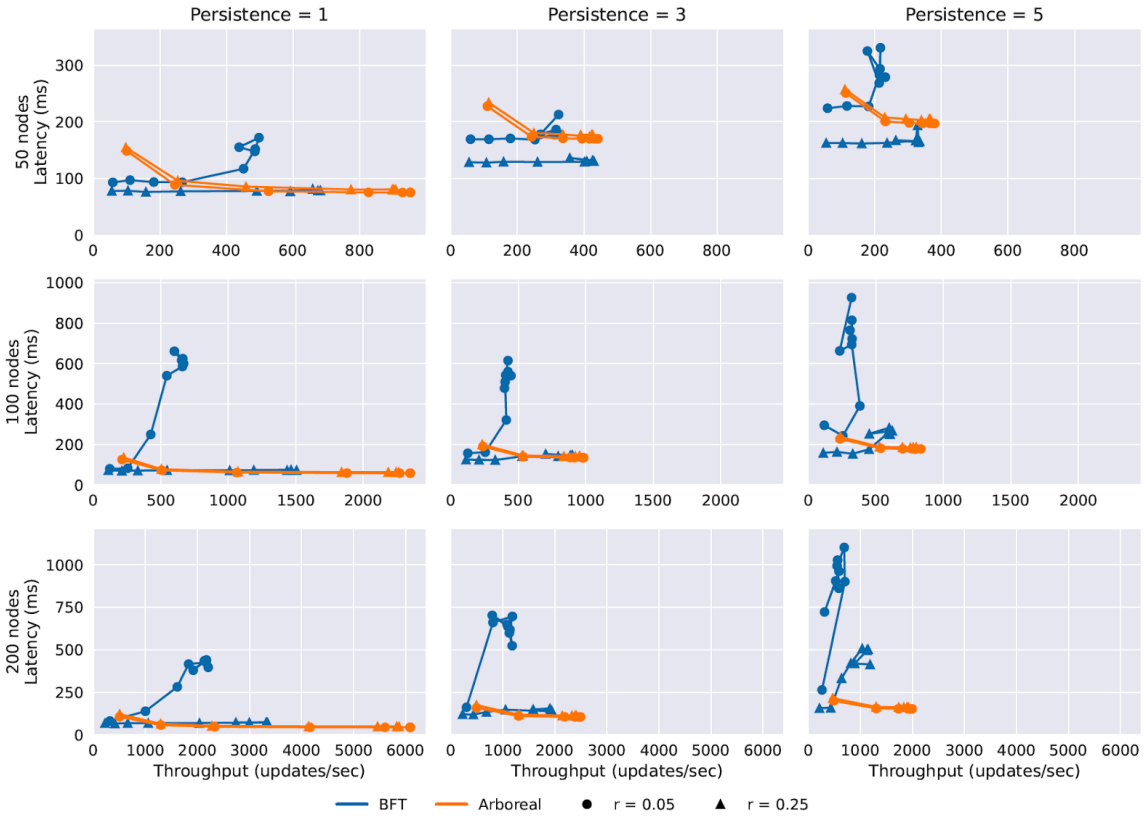

Figure 9: *Throughput–Latency Trade-offs.*

Figure 9 illustrates the relationship between throughput and average update latency for the proposed Byzantine-resilient system and Arboreal, a crash-tolerant baseline that provides comparable consistency and partial replication guarantees. Each point corresponds to a distinct update generation rate, allowing the system behaviour to be observed from light load up to saturation.

The results show that throughput increases proportionally with the update rate until a saturation point is reached, after which latency grows rapidly. The saturation threshold varies with persistence level and replication ratio, reflecting the increased processing and validation effort required under stronger durability guarantees.

As expected, the Byzantine-resilient system achieves lower maximum throughput than the crash-only baseline. Nevertheless, latency remains comparable to Arboreal up to saturation, indicating that the additional cryptographic validation and dependency tracking do not introduce prohibitive overhead under moderate load. The results also reveal that memory consumption for maintaining dependency graphs and indexes is the primary limiting factor at scale.

Visibility Time and Cross-Partition Dependencies

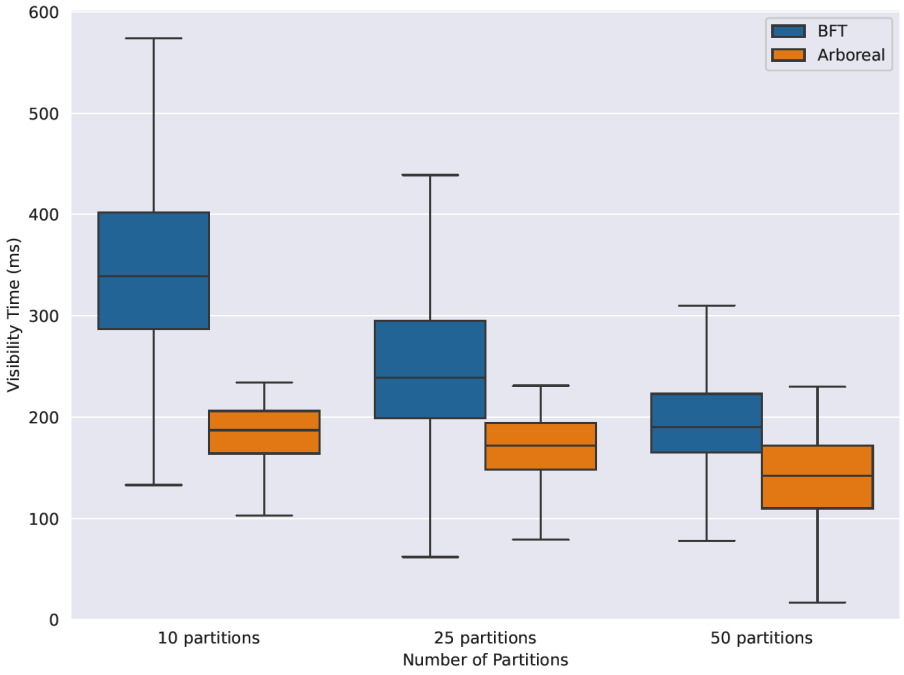

Figure 10: *Visibility Time of Updates.*

Figure 10 reports the visibility time of updates, defined as the interval between update generation and the moment it becomes visible to all relevant replicas. Visibility time captures not only dissemination delay but also the impact of causal validation and ticket processing.

The results show that visibility times in the Byzantine-resilient system exhibit higher variance than in the baseline, particularly as the number of partitions increases. This behaviour is largely explained by the reliance on tickets to validate inter-partition dependencies. Updates requiring tickets must first be processed by trusted peers before becoming visible to replicas outside the originating partition, introducing an inherent lower bound on visibility time.

While some configurations benefit from shorter overlay diameters, leading to faster propagation in isolated cases, the overall trend confirms that ticket-based validation dominates visibility latency under uniform workloads. This highlights the importance of carefully balancing partitioning strategies and ticket generation frequency in large-scale deployments.

Resilience under Crash Faults

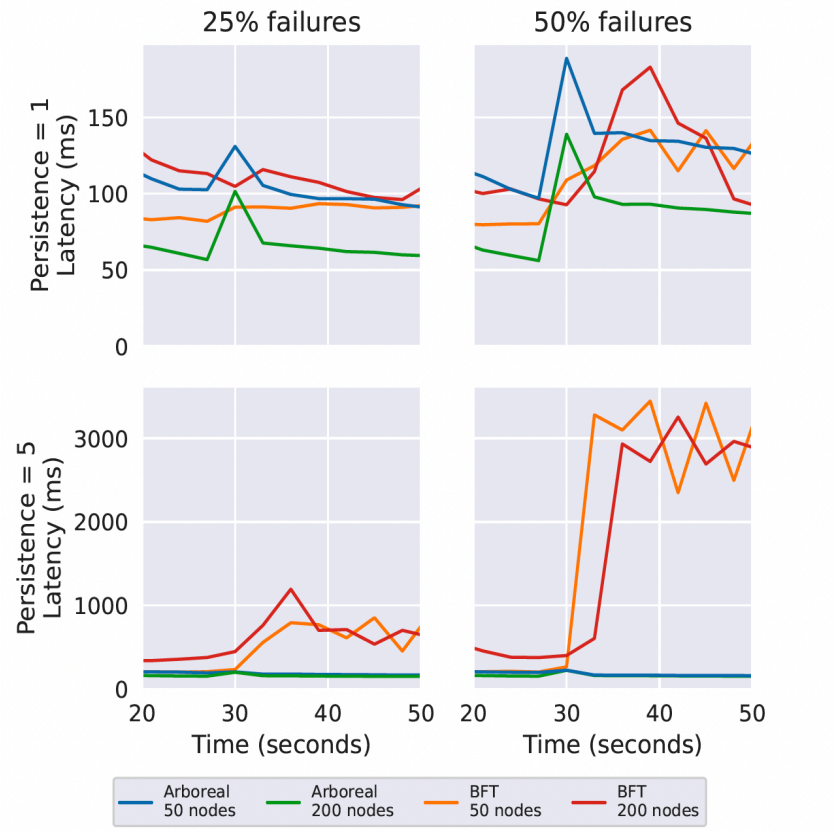

Figure 11: *Resilience under Crash Faults.*

Figure 11 examines system behaviour following the simultaneous crash of a fraction of participating nodes, reporting average update latency over time for different persistence levels and system sizes. The fault injection occurs after the system has reached a steady operational state.

The results show an immediate increase in latency following the crash, reflecting the disruption of overlay connectivity and the need for peers to re-establish stable neighbourhoods. Recovery is slower than in the crash-tolerant baseline, as the randomized overlay requires time to converge and satisfy persistence constraints.

Higher persistence levels exacerbate this effect, as nodes must obtain acknowledgments from a larger number of neighbours, sometimes relying on trusted peers to compensate for missing connections. Despite these transient effects, the system eventually stabilizes and resumes normal operation, demonstrating resilience under realistic failure scenarios.

Resilience under Byzantine Faults

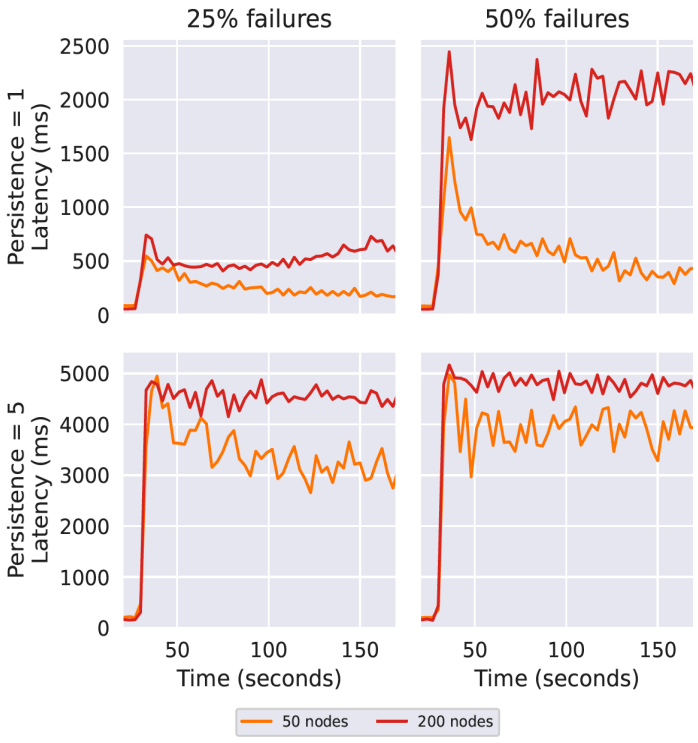

Figure 12: *Resilience under Byzantine Faults.*

Figure 12 reports the system behaviour under Byzantine faults, in which a subset of nodes actively attempts to disrupt communication by withholding messages, biasing peer samples, and repeatedly reconnecting to correct peers. Unlike crash faults, Byzantine behaviour is detected probabilistically through timeouts and anti-entropy mechanisms.

The results show a pronounced latency spike immediately after the attack's onset, as correct nodes continue to interact with malicious peers before sufficient evidence accumulates to identify the attack. Over time, nodes detect anomalous behaviour, flush compromised views, and progressively isolate Byzantine peers.

Although recovery is slower and more variable than in crash-fault scenarios, latency eventually stabilizes, and correct nodes continue to make progress. These results demonstrate that the system preserves liveness under adversarial conditions, albeit at the cost of increased transient latency, reflecting the fundamental trade-offs inherent in decentralized Byzantine fault tolerance.

Summary of Evaluation Results

Overall, the experimental evaluation demonstrates that the proposed replicated data management mechanisms provide a viable trade-off between consistency, scalability, and fault tolerance in large-scale edge environments. By avoiding global consensus and exploiting partial replication and causal consistency, the system remains performant under benign conditions while preserving robustness to Byzantine faults. These results support the approach's suitability for decentralised edge scenarios targeted by TaRDIS, where trust assumptions are weak and operational conditions are highly variable.

4.4.7 Limitations and Lessons Learned

While the results demonstrate the feasibility of Byzantine-resilient replication at the edge, several limitations remain. Metadata overhead associated with causal tracking grows with system size and update concurrency, requiring careful tuning in large deployments. Partial replication introduces additional complexity in replica selection and data placement, which may benefit from adaptive strategies informed by runtime monitoring.
More broadly, the work highlights the difficulty of balancing security and performance in decentralised edge systems. Stronger fault models inevitably incur additional costs, and the appropriateness of such mechanisms depends on application requirements and threat assumptions. These lessons informed subsequent WP6 work on observability and adaptive management and point to promising directions for future research beyond the scope of TaRDIS.

4.5 Dynamic and Flexible Replication for Swarm-Based Applications

4.5.1 Context and Motivation

Modern distributed applications increasingly operate in highly dynamic, infrastructure-poor environments, such as mobile swarms, collaborative edge deployments, vehicular networks, and intermittently connected IoT systems. In these settings, nodes are expected to operate autonomously, tolerate disconnections, and continue to provide local functionality despite the absence of stable connectivity or centralized coordination. Traditional data management solutions, however, are typically designed around fixed infrastructures, centralized services, or tightly coupled client–server architectures, which limit their applicability in such environments.

Local-first data management approaches partially address these challenges by allowing clients to operate on local replicas and synchronize with a backend when connectivity permits. While effective for small-to medium-scale deployments, these systems usually rely on centralized servers to reconcile state and mediate convergence. As a result, their scalability, fault tolerance, and convergence behaviour are constrained by the availability and performance of the backend infrastructure.

Nimbus targets a different point in the design space. It is designed as a fully decentralized data management system that operates without any dedicated infrastructure, enabling peer-to-peer interaction among autonomous nodes.

4.5.2 Objective and Design Goals

A key motivation behind Nimbus is the need to decouple data availability and consistency from centralized coordination, while still supporting expressive data models and flexible access policies. By structuring data into fine-grained information units and allowing replication sets to evolve dynamically, Nimbus enables applications to balance storage, communication, and consistency requirements in line with their operational constraints. Moreover, its modular architecture allows different membership, control, and replication protocols to be plugged in, making Nimbus adaptable to a wide range of deployment scenarios.

As such, Nimbus introduces a decentralized data management system designed for swarm systems and highly dynamic environments. Nimbus was designed to operate without any dedicated infrastructure by enabling fully peer-to-peer interactions among clients, with no centralized components. Nimbus was first introduced in D6.2, presenting the system's overall architecture and design while it was still under development. Since then, Nimbus has been finalized and has evolved to accommodate additional features and functionalities, such as improved partial replication, modular components, labelled data, and others, as detailed in the following sections.

4.5.3 Technical Description

Nimbus is a decentralized data management system designed to operate without any centralized infrastructure. Each node functions as an independent replica, capable of performing reads, writes, and deletes locally—even when disconnected—and synchronizing with peers opportunistically. Its architecture is modular, offering pluggable control and replication components that allow Nimbus to adapt to different network conditions and application needs. Nimbus supports partial dynamic replication of its *information units*, the system's fundamental replication units. Internally, Nimbus leverages Conflict-free Replicated Data Types (CRDTs) to model its data and replication protocols that guarantee strong eventual consistency across replicas. Applications interact with the Nimbus datastore by issuing operations on abstract data structures—such as sets, maps, flags, and others—which are internally mapped to CRDTs and associated with distinct information units.

Flexibility is a core design principle of Nimbus, which provides modularity across its components, as illustrated in the following diagram:

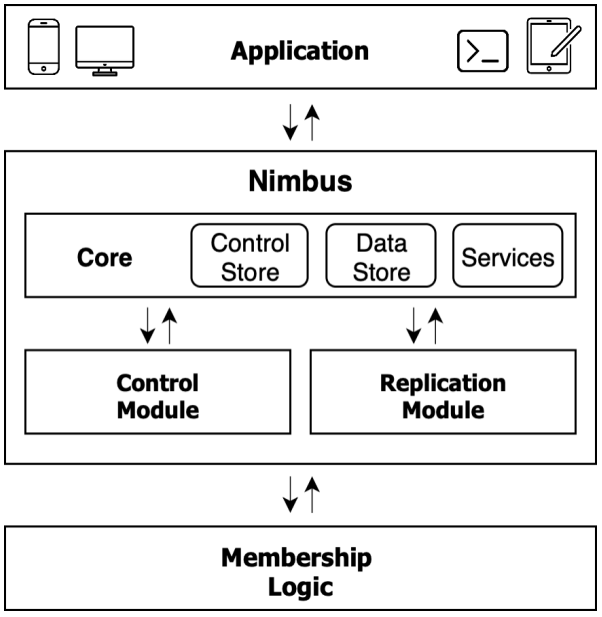

Figure 13: *Nimbus Architecture.*

With this in mind, Nimbus architecture is divided into different components (see Figure 13):

- **Core:** This layer encapsulates the majority of Nimbus's functionality and integrates all components. It is primarily responsible for interacting with client applications and maintaining both control information and application data. To accomplish these tasks, the core communicates with the control and replication modules, which handle coordination and communication with other nodes. Additionally, the core provides a set of services that support various Nimbus functionalities, including persistence and filtering.
- **Control Module:** The control module is a pluggable component responsible for synchronizing the system's control information. It interfaces directly with the core and receives updates from the membership layer, similarly to the replication module. Its main task is to ensure that control information is consistently propagated throughout the system.
- **Replication Module:** The replication module synchronizes the storage data—specifically, the CRDTs maintained in the core's Data Store. Like the control module, it is also pluggable and can be replaced with different propagation protocols to better suit application requirements or to enable more efficient synchronization in particular scenarios.

Data Model

Nimbus adopts a hierarchical key–value data model in which data is organized into keyspaces, collections, and objects. A collection represents a logical unit of information and contains objects implemented as CRDT-backed replicated data structures (e.g., counters, registers, sets, maps). This design enables nesting and composition of data, allowing complex structures to be represented naturally.

Each object is uniquely identified by the path from its collection to its embedded data structure (see Figure 14), supporting fine-grained interaction and selective replication under the various information units.

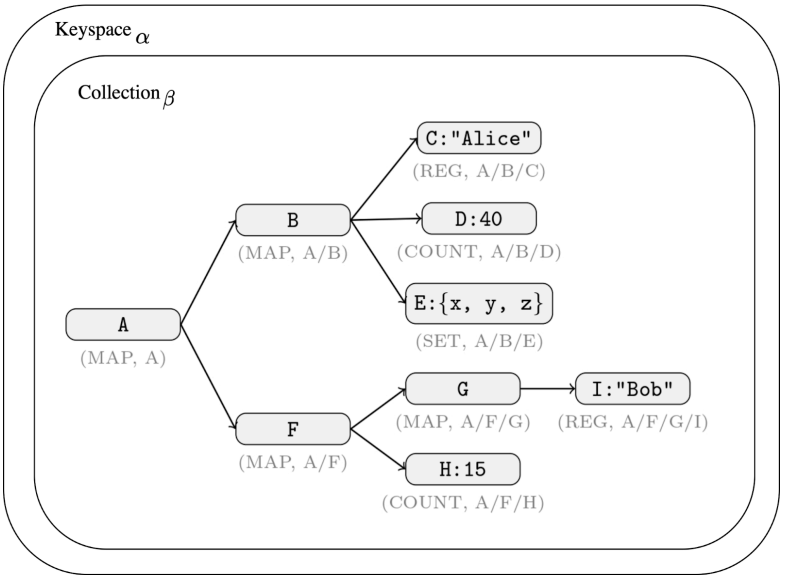

Figure 14: *Embedded Data Structure.*

Since D6.2, Nimbus data model has evolved to accommodate optional tags that can be attached to collections or objects to influence *storage behaviour*, with options such as timed entries for devices with limited resources; *propagation strategies*, by relying on priority lists for message delivering and geographical position on objects; and *query filtering* for removing expired entries or filter result fetching by location.
Developers can use these three mechanisms to design and implement their own propagation and replication protocols that best suit their needs, as described in the following sections.

Clients create objects by supplying their complete identifier, specifying the desired data structure type, and optionally providing an initial value. Deletion of an object follows an analogous procedure. Applications access objects by invoking operations on their replicated data structures. For example, after creating a counter, an application may execute one of the counter available operations — $COUNTER\_GET$, $COUNTER\_INC$, and $COUNTER\_DEC$. Developers select the appropriate replicated structure for their data, and Nimbus transparently handles synchronization across replicas in accordance with the semantics of the underlying CRDTs. Clients may issue operations while offline, ensuring high availability. Nimbus guarantees that updates are eventually propagated to all replicas, respecting the issue order, preventing rollbacks, and ensuring strong-eventual convergence.

Control Module

Control information, or metadata, describes the available information units and their state. Unlike storage data, which relies on partial replication—allowing each node to maintain only the subset of data it is interested in or responsible for—control information is propagated to all nodes. This ensures that every participant in the system has a consistent and up-to-date view of the system's structure and metadata. By maintaining this small but critical metadata fully replicated, Nimbus enables coherent decision-making at each node without requiring centralised coordination or synchronous communication.

Information units serve as the replication units in Nimbus, storing content that supports its functionalities. These units are organized hierarchically into keyspaces and collections, serving as a mechanism for structuring data and replication. When an information unit is created, the application specifies its configuration parameters, including the owner (i.e., the creating node), access and replication policies, the set of authorized nodes, and its replica set. It is also possible to modify certain parameters of the information unit, such as the authorization set, to revoke permissions or add them to new nodes. Nimbus also supports dynamic replication by allowing modification of the replica set for each information unit.

Since collections hold storage data, they also maintain object identifiers and optional tags, a feature developed after D6.2. Tags enhance Nimbus collections and objects with additional, optional behaviour, divided into three categories: storage tags, propagation tags, and query tags. Storage tags affect storage dimensions and access; propagation tags may influence the replication module by encoding properties such as geographical location or priorities; and, finally, query tags enable filtering during queries to the system, e.g., to exclude stale entries or data outside a certain region. Tags are optional and transparently managed by Nimbus during execution.

Control information is stored and validated in Nimbus's control store but disseminated via its control module - a pluggable component responsible for synchronizing control information with each node's neighbourhood, as defined by the external (also pluggable) membership logic. To efficiently track and verify the state of information units, Nimbus leverages adapted Merkle trees.

Nimbus prototype offers various implementations of this module. For instance, a gossip-based protocol with anti-entropy is provided, where each node communicates randomly with a subset of its neighbours to disseminate control information.

Replication Module

As discussed earlier, to meet the demands of local-first applications—particularly in dynamic environments where nodes cannot be expected to have sufficient resources to replicate the entire application dataset—Nimbus employs a dynamic partial-replication model.

In this design, information units serve as fine-grained replication units, each maintaining its own replication set. The nodes maintain only the keyspaces and collections they are configured to handle, and each device keeps active connections only to some of its neighbours, illustrating a partially connected, decentralised topology without any apparent structure:

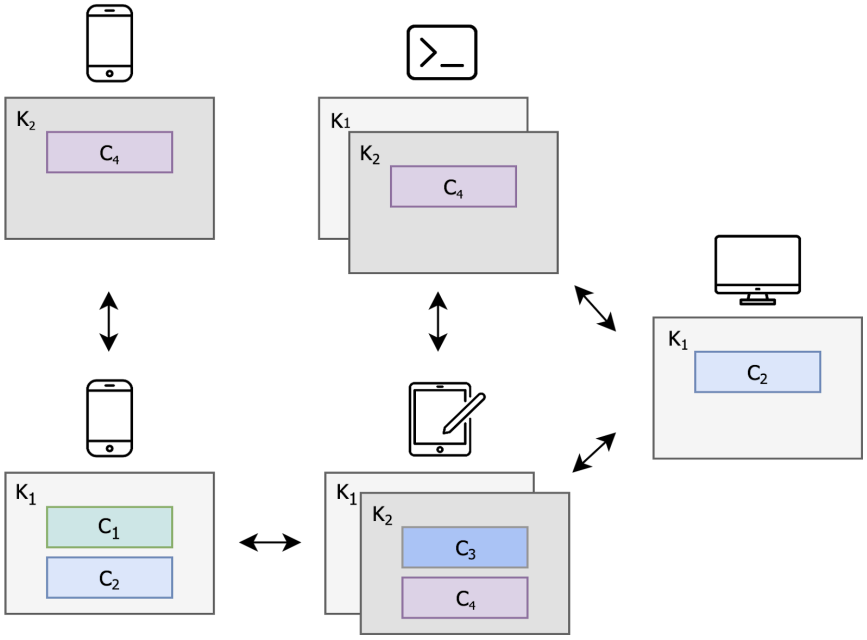

Figure 15: *Nimbus Synchronization Mechanism.*

To address this challenge, Nimbus adopts an explicit replication policy in which replicas are managed by nodes with access to the corresponding information units. Authorised nodes are free to add or remove replicas for any information unit. When a node is added as a replica, an initial synchronization step transfers the relevant data to it. Conversely, when a node is removed from the replication pool - or exits the system gracefully - the replication information is removed, and the control information is updated accordingly. In the case of a node failure or temporary disconnection, since information is replicated

across multiple nodes and no additional coordination is required, data remains available and accessible.

Under this design, where nodes maintain partial replicas of storage data alongside global knowledge of control information, state-based CRDTs - specifically, delta-based variants - offer a natural and efficient synchronization mechanism (as illustrated in Figure 15). This is particularly advantageous in decentralized, peer-to-peer settings with intermittent connectivity, as it minimizes communication costs while still guaranteeing strong eventual consistency and convergence. Nimbus uses an adapted and optimized version of δ-CRDTs to better fit its data model.

In Nimbus, even nodes that do not replicate an information unit can still access it, provided they satisfy its access policy. To support this, each node maintains a local set of operational policies, which may be updated by reconfiguring the information unit parameters. These policies govern how writes, reads, and deletes behave when a node is authorized to interact with an information unit but is not permitted to replicate it.

Nimbus supports a range of policies to accommodate different usage scenarios, offering varying degrees of restrictiveness to clients: The most restrictive, $LOCAL\_OP$, permits access only if the data is already available locally (i.e., the node replicates it); $REMOTE\_OP$ executes the operation remotely on a node that replicates the information unit and returns the result; finally, the most permissive, $REPLICATE\_OP$, requests to begin replicating the information unit, thereby updating the replication pool and fetching the data before committing the operation. Additionally, for reads, Nimbus provides an optional caching policy, $CACHE\_OP$, whereby remote reads are cached locally for a configurable duration, avoiding repeated remote operations for frequently accessed objects.

Similar to control information, storage data is maintained in Nimbus's core data store but propagated using the replication module. The replication module is a pluggable component that synchronizes data (i.e., CRDT states) between replicas of each information unit. It interacts with the external membership module to obtain information about the current neighbourhood, which it uses to build partial overlays for each replication unit. Since this component is pluggable, Nimbus defines a common interface for it, supporting the following functions: propagating and receiving data (objects and their associated tags); issuing remote operations and requesting remote states; updating the set of replicas for each information unit; and collecting metrics (e.g., hop count, message size) about exchanged objects. Moreover, the replication modules can use the information unit propagation tags (described in the previous sections) to enable more efficient and tailored propagation protocols across diverse scenarios.

This design embodies Nimbus's philosophy of adaptability and modularity, allowing nodes to operate with only the data they need while supporting flexible interaction patterns under diverse access policies.

Core Module

At the heart of Nimbus lies the core module, which provides the fundamental services for managing, storing, and synchronizing data across the system. The core module encapsulates the logic for maintaining the control and data stores, enforcing access and replication policies, and delivering the diverse functionalities that Nimbus offers through

auxiliary services. It is designed to tolerate node failures and network partitions, ensuring eventual consistency and seamless recovery once connectivity is restored.

The core interacts directly with the application layer via a well-defined, event-based interface. Clients interact with Nimbus by issuing requests (e.g., creating a keyspace, updating an object) and receiving appropriate replies (e.g., status of an update, the value of an object). Object reads can be returned in different formats, including JSON or raw byte payloads. To support real-time applications, Nimbus also provides reactive mechanisms to notify clients when new information becomes available from other nodes—for example, upon object updates or when accessing newly replicated information units.

Nimbus also supports a rich set of configuration parameters that allow fine-grained control over its execution behaviour. These include persistence policies (e.g., immediate persistence, batched writes, or deferred persistence), as well as configurable time-to-live (TTL) values for multiple internal timers, such as cache expiration, metadata synchronisation intervals, and replication rounds. By tuning these parameters, deployments can balance durability, performance, resource consumption, and freshness guarantees to better match the characteristics of the target environment and application workload.

The following subsections describe in detail the main components of Nimbus's core.

Control & Data Store

The control store and data store form the foundation of Nimbus's internal state management, maintaining the metadata and application data required for operation. Each of these stores integrates with its respective module, ensuring that information is propagated to other nodes according to the chosen implementation.

The control store provides a consistent view of the system's organization and replication policies across nodes, allowing nodes to access the most up-to-date system state before issuing critical operations. Conversely, the data store holds the actual application objects and their replicated state—namely, the CRDTs—organized into the same hierarchy of information units defined in the control store. In addition, the data store maintains, validates, and updates (when necessary) the various tags associated with each collection.

Both stores reside in memory for efficient access but can also be periodically persisted to disk for increased resilience against failures

Generic Services

Beyond its core control and data management components, Nimbus provides a set of auxiliary services that enhance functionality and usability in real-world deployments. These services address practical concerns such as durability through disk persistence, caching, recovery of pending operations, and advanced querying via tags.

Nimbus supports disk persistence at the level of its information units. Both control and data information are written to disk and can later be retrieved in the event of a node

failure. Nimbus leverages its hierarchical data structure to perform efficient writes and reads, using a custom engine that reuses space from deleted or garbage-collected objects to store updates or expand capacity when needed.

Applications & Use-Cases

Nimbus supports a broad range of decentralised and local-first applications thanks to its composable data model and adaptive replication capabilities. In collaborative scenarios, such as decentralised presentation evaluation, Nimbus allows each participant to replicate only the presentations they access while ensuring that feedback and ratings converge across relevant nodes.

Moreover, in communication-constrained environments—such as satellite constellations or vehicular networks—replicated states can be exchanged opportunistically, allowing nodes to refine navigation parameters or disseminate alerts without reliance on cloud services. Nimbus is equally applicable in resource-limited or infrastructure-free contexts, including IoT deployments, emergency response systems, and mobile swarms, where autonomous operation and dynamic peer-to-peer synchronisation are essential.

Different applications and demos for Nimbus are available under the Nimbus Applications repository[40], as well as the generic micro-benchmark used for evaluation [41].

4.5.4 Relationship with Other WP6 Components

Nimbus was designed with flexibility and adaptability as primary goals, allowing it to operate across a wide range of deployment environments and system configurations. Consequently, the system can be readily integrated with the different solutions presented in this deliverable for membership management, communication, and metric collection and aggregation, without requiring changes to its core architecture. Nimbus builds directly on the communication and coordination mechanisms provided by the Babel ecosystem, leveraging the protocols and services offered by the framework for membership management, message dissemination, and coordination. In addition, Nimbus exposes an interface that closely follows the APIs defined by WP6 for data management and communication, enabling the seamless integration of pluggable solutions for the various components of the TaRDIS toolbox.

Synchronizing state in replicated systems is one of the most challenging aspects of distributed computing. In such systems, multiple copies of data are maintained across different nodes to ensure high availability and fault tolerance. However, ensuring that all replicas remain consistent and reflect the same state at any given time is difficult due to network latency, failures, and the inherent asynchrony of distributed environments. Conflict-Free Replicated Data Types (CRDTs) address this challenge by allowing replicas to be updated independently and concurrently, while guaranteeing that all replicas eventually converge to the same state, even in the presence of network partitions or failures. The TaRDIS toolbox, as already described in D6.2, includes an extensible and serializable CRDT library integrated with the Babel ecosystem. Since the publication of

[40] https://codelab.fct.unl.pt/di/research/tardis/wp6/internal-tools/nimbus/examples/sample-applications

[41] https://codelab.fct.unl.pt/di/research/tardis/wp6/internal-tools/nimbus/evaluation

D6.2, this library has been extended with additional data structures (e.g., flags and matrices) to support a broader range of application requirements, as well as with multiple propagation protocols that enable the correct and efficient use of these CRDTs. These data structures and protocols are leveraged by the Nimbus prototype.

Finally, the evaluation of Nimbus was conducted using the Network Emulation and Experimental Infrastructure contribution described in Section 4.9.

4.5.5 Advances Beyond the State-of-the-Art

Replicated data systems aim to ensure high availability, fault tolerance, and low latency by maintaining copies of data across multiple nodes.

SwiftCloud[42] is a centralized geo-replicated cloud storage system that leverages CRDTs to provide low-latency, highly available operations on replicated data while maintaining eventual consistency. Various systems have explored decentralized architectures, distributing control and storage over networks of nodes, typically organized in unstructured or structured overlays and propagating updates via message passing. While these approaches successfully decentralize control and storage, they often assume full replication, immutable data models, or dedicated infrastructure, limiting adaptability in dynamic and resource-constrained environments[43]. Multiple systems that replicate data on the client have been proposed in the past. Bayou[44] introduced a fully replicated, peer-to-peer storage system where all replicas could accept updates independently—even while disconnected—and later reconcile them using a pairwise anti-entropy protocol. Conversely, Cimbiosys [45] focused on selective replication, allowing nodes to maintain only a subset of data relevant to them by defining static constraints. Nimbus improves upon these by supporting dynamic partial replication with strong eventual consistency via CRDTs. Unlike Bayou's full replication and custom conflict resolution or Cimbiosys's static constraints, Nimbus provides a flexible and adaptable architecture suited for modern decentralized, local-first environments. As such, CRDTs provide a natural foundation for building decentralized data systems that support concurrent updates without coordination, guaranteeing convergence across replicas. GUN[46] is a decentralized, graph-based data store designed for peer-to-peer environments with eventual consistency guarantees. GUN uses a last- write- wins strategy and timestamped updates, supporting CRDT - like behaviours for some data types but lacking a full formal CRDT framework with guarantees for complex structures.

---

[42] Marek Zawirski, Annette Bieniusa, Valter Balegas, Sérgio Duarte, Carlos Baquero, Marc Shapiro, and Nuno M. Preguiça. 2013. SwiftCloud: Fault-Tolerant Geo-Replication Integrated all the Way to the Client Machine. CoRR abs/1310.3107 (2013). arXiv:1310.3107 http://arxiv.org/abs/1310.3107

[43] https://couchdb.apache.org/

[44] D. B. Terry, M. M. Theimer, Karin Petersen, A. J. Demers, M. J. Spreitzer, and C. H. Hauser. 1995. Managing update conflicts in Bayou, a weakly connected replicated storage system. SIGOPS Oper. Syst. Rev. 29, 5 (Dec. 1995), 172–182. https://doi.org/10.1145/224057.224070

[45] Venugopalan Ramasubramanian, Thomas L. Rodeheffer, Douglas B. Terry, Meg Walraed-Sullivan, Ted Wobber, Catherine C. Marshall, and Amin Vahdat. 2009. Cimbiosys: a platform for content-based partial replication. In Proceedings of the 6th USENIX Symposium on Networked Systems Design and Implementation (NSDI'09). USENIX Association, USA, 261–276.

[46] https://gun.eco/

On the other hand, Legion[47] is a peer-to-peer framework enabling direct browser-to-browser interactions, building on Δ-CRDTs to share and synchronize mutable, replicated documents. However, Legion relies on centralized services for some of its functionality and lacks Nimbus's extensible data model.

OrbitDB[48] is a serverless, distributed, peer-to-peer database using IPFS[49] as its storage layer and Merkle-CRDTs for merging updates. OrbitDB guarantees immutability and verifiability by maintaining an append-only log, which, while transparent, limits flexibility. Nimbus shares some goals and design choices with OrbitDB, but is specifically designed to support the dynamism and scale required by swarm- and local-first applications, with a more flexible approach for diverse environments.

4.5.6 Experimental Validation and Evaluation

In this section, we study the performance of the Nimbus prototype and compare it against a CouchDB + PouchDB deployment. For evaluation, we developed a generic anti-entropy protocol for Nimbus based on gossip communication that establishes partial overlays per information unit across different instances. The protocol relies on periodic, configurable synchronization rounds to disseminate the deltas accumulated by each replica, enabling updates to propagate gradually through the system despite partial replication and connectivity.

We compare Nimbus against a CouchDB + PouchDB deployment, representing a hybrid client–server architecture in which client nodes maintain local replicas for offline reads and updates and periodically synchronise their state with a corresponding CouchDB server. In this setup, PouchDB provides local persistence and client-side conflict handling, while CouchDB serves as the persistent backend that reconciles divergent replicas during synchronisation.

Experimental Setup

All experiments were conducted on a cluster of eight machines, each equipped with two AMD EPYC 7343 16-core processors and 256 GB of RAM. Instances were evenly distributed across the available machines.

To scale the experiments to a large number of nodes, we relied on the Network Emulation and Experimental Infrastructure tool described in Section 4.9. To emulate geographic distribution, the infrastructure introduces network latency between nodes based on their Euclidean distance, with a maximum end-to-end latency of 150 ms.

All experiments were executed using Docker containers deployed across the available machines.

[47] Albert van der Linde, Pedro Fouto, João Leitão, Nuno Preguiça, Santiago Castiñeira, and Annette Bieniusa. 2017. Legion: Enriching Internet Services with Peer-to-Peer Interactions. In Proceedings of the 26th International Conference on World Wide Web (Perth, Australia) (WWW '17).
[48] https://orbitdb.org/
[49] https://ipfs.tech/

Deployment Configurations

We present results for a deployment of 1000 nodes, representative of a realistic swarm-like scenario. In the Nimbus configuration, each node runs a Nimbus instance and maintains only a partial replica of the overall replication set. Membership among Nimbus instances is established and maintained using a modified version of the HyParView[50] protocol. Each node periodically propagates its accumulated deltas every 6 seconds to a subset of its neighbours.

In contrast, the CouchDB + PouchDB deployment dedicates three machines to host CouchDB server instances, while the remaining machines emulate client nodes running PouchDB instances. Each client periodically synchronises its local replica with its corresponding CouchDB server using the same synchronisation interval as Nimbus.

Metrics

We evaluate Nimbus using three primary metrics: staleness, convergence time, and hop count. Staleness captures the average number of missing operations per node during execution. Convergence time measures how long it takes for an update to become visible at all relevant replicas. Hop count represents the number of intermediate nodes a delta passes through before reaching its destination.

To provide a meaningful comparison, we collect analogous client-observable metrics for CouchDB + PouchDB. In this setting, staleness is measured as the divergence between a client's local replica and its corresponding CouchDB instance at read and write time. Convergence time is defined as the elapsed time between issuing a local update and observing the same revision at the server. Additionally, we record the number of detected conflicts during read operations as an indicator of inconsistency exposed to applications.

Results and Discussion

We now present and briefly discuss the main experimental results we have obtained for this contribution.

[50] J. Leitão, J. Pereira, and L. Rodrigues. "HyParView: A Membership Protocol for Reliable Gossip-Based Broadcast." In *Proceedings of the 37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN 2007)*, Edinburgh, United Kingdom, June 2007, pp. 419–429. IEEE Computer Society.

Convergence Time

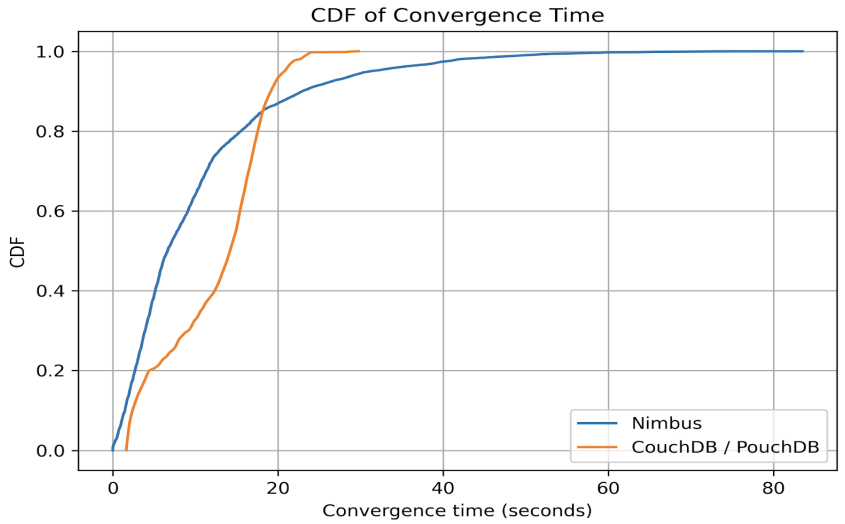

Figure 16: *Cumulative Distribution Function of convergence Time.*

Figure 16 presents the cumulative distribution function (CDF) of convergence time for both Nimbus and the CouchDB/PouchDB deployment. The results highlight a fundamental trade-off between decentralized, gossip-based synchronization and centralized, client–server replication architectures.

CouchDB/PouchDB exhibits faster overall convergence, with nearly all updates propagating within approximately 20 seconds. The steep slope and early saturation of the CDF reflect the efficiency of its hybrid architecture, in which updates follow a direct two-hop path (client → server → client). This predictable topology minimizes the number of synchronization steps required for convergence and enables rapid dissemination once updates reach the server.

In contrast, Nimbus shows a more gradual convergence profile, with the distribution spanning a wider temporal range. While the majority of updates converge within a similar time window (around twenty seconds), a non-negligible fraction requires longer to reach all relevant replicas. This behaviour is a direct consequence of Nimbus's fully decentralized design, in which updates propagate through multiple intermediate replicas before reaching the entire replication set.

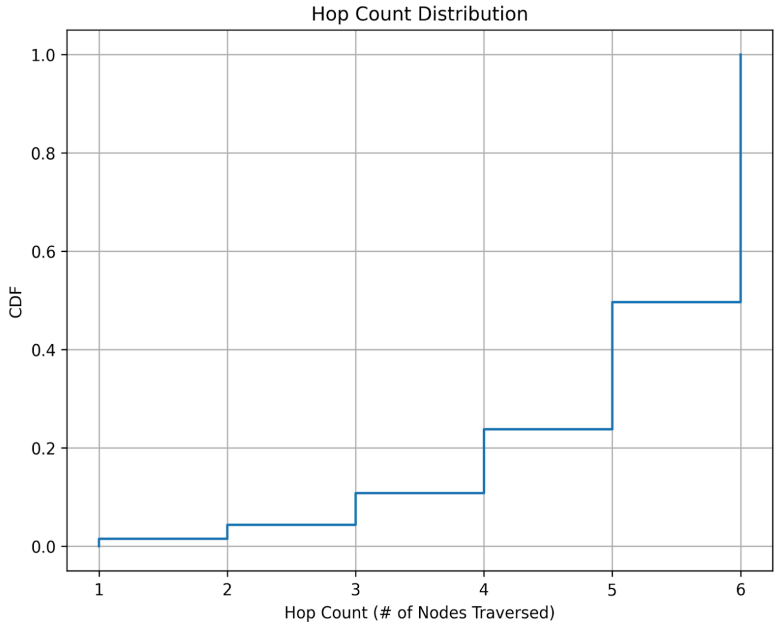

Figure 17: *Nimbus Hop-Count Distribution.*

The hop-count distribution shown in Figure 17 provides additional context for Nimbus's convergence behaviour. Most updates are propagated within four to five hops, although some traverse up to six intermediate nodes. Given the six-second synchronization interval used in the experiments, this hop-count distribution closely matches the observed convergence times. The variability in hop count—ranging from one to six hops—directly contributes to the wider spread of the convergence-time CDF.

Despite these longer convergence times, Nimbus remains well-suited for swarm-like and infrastructure-less environments, where eventual consistency is sufficient and reliance on centralized services is undesirable or infeasible. The overhead introduced by multi-hop gossip propagation represents the inherent cost of achieving decentralization, fault tolerance, and infrastructure independence.

Staleness

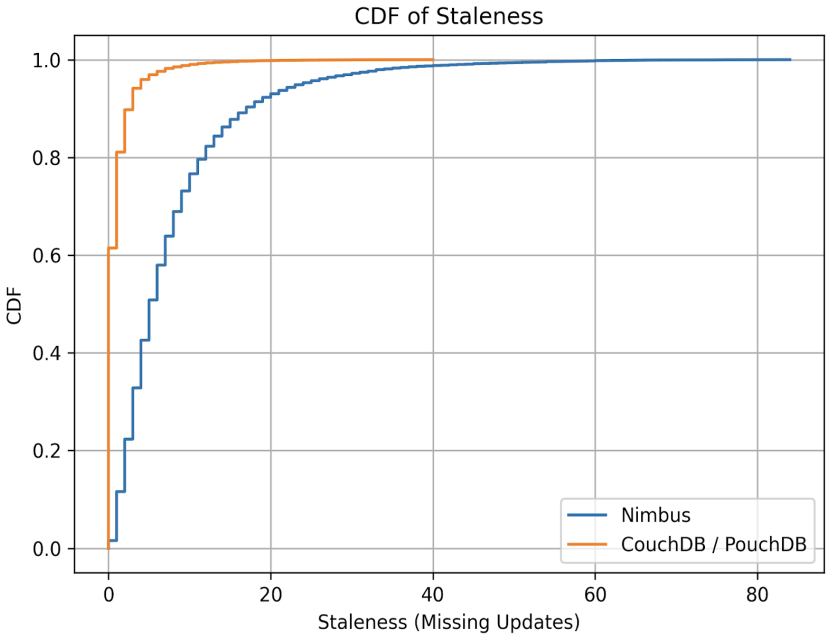

Figure 18: *Nimbus: Accumulative Distribution Function of Staleness.*

Figure 18 presents the cumulative distribution function of staleness, measured as the number of missing updates observed by a node at read time, for both Nimbus and CouchDB/PouchDB. CouchDB/PouchDB exhibits consistently low staleness, with nearly all observations showing fewer than five missing updates. The steep CDF and early saturation reflect the system's hybrid client–server synchronization model, in which clients frequently synchronize with authoritative servers, keeping replicas closely aligned with the latest state.

This low divergence, however, comes at the cost of requiring persistent server availability and stable client–server connectivity.

Nimbus, by contrast, exhibits higher staleness, with the distribution extending across a broader range. Approximately 50% of observations report fewer than 10 missing updates, while the median staleness is between 12 and 15 operations. This increased divergence is an inherent property of gossip-based dissemination over partial overlays, where updates are propagated incrementally across multiple synchronization rounds rather than immediately pushed from a central authority.

Funded by the European Union

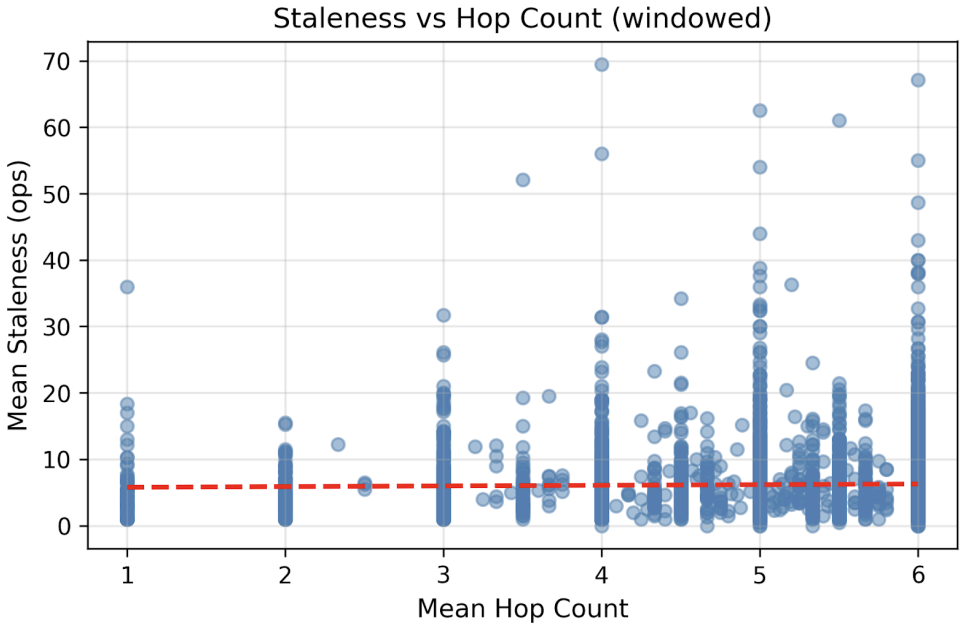

Figure 19

Figure 19 provides an insight on the relationship between staleness and network topology in Nimbus through observation windows. The scatter plot shows that staleness is largely independent of hop count, with observations uniformly distributed across different propagation distances. The red dashed line, representing the mean staleness (approximately six operations), remains stable across all hop-count values.

This indicates that Nimbus's staleness is not primarily driven by propagation distance or topological position within the overlay, but instead by the probabilistic nature of gossip dissemination and the fixed synchronization interval. Importantly, the absence of a strong correlation between hop count and staleness suggests that Nimbus's anti-entropy protocol effectively masks topological complexity, providing relatively uniform consistency guarantees across the system.

While Nimbus exhibits higher absolute staleness than CouchDB/PouchDB, this trade-off enables decentralized operation, partition tolerance, and scalability to 1,000 nodes without centralized coordination. For applications designed around eventual consistency semantics, this level of divergence remains acceptable—particularly given that staleness does not compound with distance in the overlay.

Funded by the European Union

Conflicts and Resolution Strategies

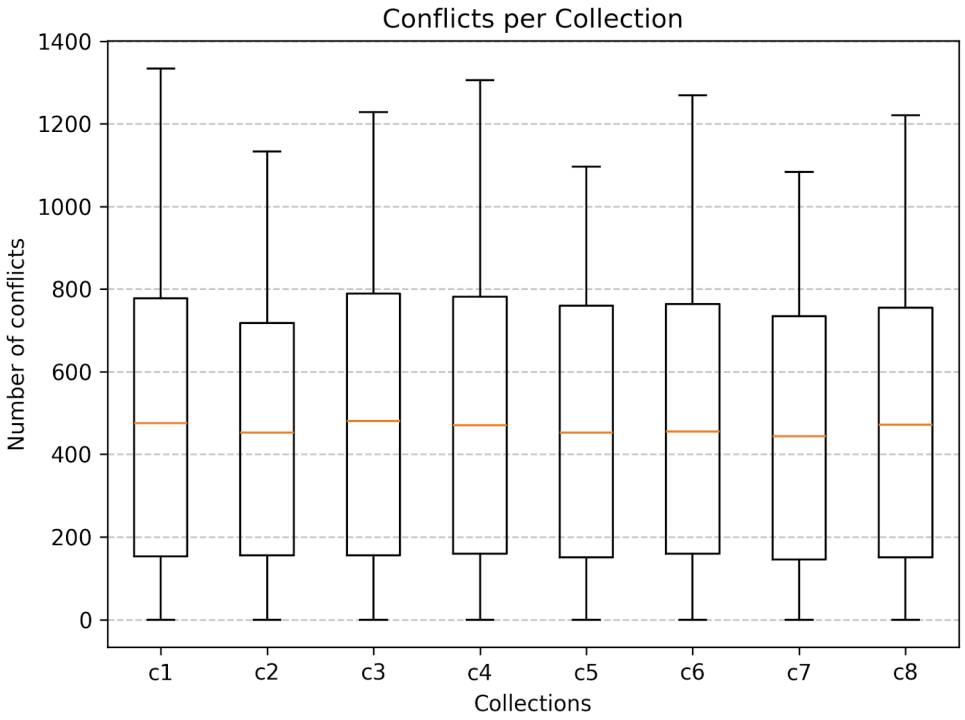

Figure 20

Figure 20 shows the distribution of conflicts detected across several collections (i.e., replication units) in the CouchDB/PouchDB deployment. The box plots reveal a high number of conflicts across all eight collections, with median values consistently between 450 and 470 conflicts per collection. This behavior is characteristic of CouchDB's multi-version concurrency control model, in which concurrent updates to the same document at different clients result in conflicting revisions that must be resolved by the application or by the system.

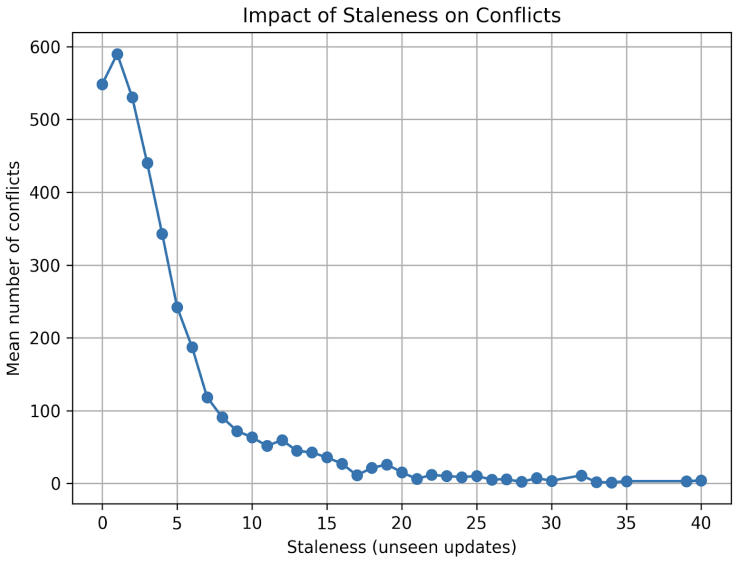

Figure 21: *Impact of Staleness on Conflicts.*

Figure 21 provides further insight into the relationship between staleness and conflict occurrence in CouchDB/PouchDB. The plot reveals a strong negative correlation: as staleness increases from zero to approximately ten unseen updates, the mean number of detected conflicts drops sharply—from nearly 600 to around 60.

This pattern exposes an important limitation of the CouchDB/PouchDB model. Conflicts are only detected when concurrent updates are observed before the server selects a canonical revision. Clients with low staleness—i.e., those synchronizing frequently—are more likely to encounter conflicts because they observe concurrent updates before resolution. Conversely, clients with higher staleness often synchronize after conflicts have already been resolved on the server, at which point the lost revisions have been discarded, and conflicts are no longer visible.

As a result, lower observed conflict rates do not necessarily indicate improved consistency; instead, they reflect the silent loss of concurrent updates.

In contrast, Nimbus experiences zero conflicts and zero data loss by design. This is a direct consequence of its CRDT-based data model, in which all concurrent updates are guaranteed to converge to the same state without conflict detection or resolution. Unlike CouchDB's revision-tree or last-write-wins strategies—which necessarily discard information when resolving conflicts—Nimbus preserves all updates issued by all replicas.

This distinction is particularly relevant for swarm-like deployments characterized by intermittent connectivity and offline operation. Applications built on CouchDB/PouchDB must implement explicit conflict-handling logic and accept that some user actions may be silently overwritten. Nimbus, by contrast, provides strong eventual consistency guarantees in which every update contributes to the final converged state—an essential property for collaborative and offline-first applications.

Evaluation Summary:

Our evaluation highlights the contrasting design trade-offs between Nimbus's fully decentralized, gossip-based architecture and the hybrid client–server model of CouchDB/PouchDB.

CouchDB/PouchDB achieves faster convergence and lower staleness by relying on direct client–server synchronization while providing offline access through client caching, resulting in predictable and tightly coupled replicas. However, this comes at the cost of centralized infrastructure dependence and a conflict-resolution model that may silently discard concurrent updates.

Nimbus, in contrast, prioritizes decentralization, infrastructure independence, and fault tolerance. While its gossip-based anti-entropy protocol leads to higher convergence times and increased staleness, these effects remain bounded and scale well to large deployments. Importantly, staleness in Nimbus is largely independent of network topology, indicating that its partial overlays and synchronization strategy effectively mask propagation distance.

Most notably, Nimbus eliminates conflicts and data loss entirely through its CRDT-based data model, guaranteeing that all concurrent updates are preserved and eventually converge. This property is critical for swarm-like, collaborative, and offline-first environments, where intermittent connectivity and autonomous operation are the norm.

Overall, the results demonstrate that Nimbus offers a robust and scalable alternative to centralized data-management systems, trading faster convergence for strong eventual consistency, resilience, and true peer-to-peer operation—qualities essential for dynamic and infrastructure-less deployments.

Repository

More details about the API provided by Nimbus can be found in D3.5, in the Babel Protocol Commons under the storage package, and on the TaRDIS wiki. The prototype and evaluation for Nimbus are available on the Nimbus Git repository[51], as well as its wiki, which provides extended information regarding the repository structure, Nimbus's underlying architecture, and instructions for execution.

4.5.7 Limitations and Lessons Learned

Nimbus demonstrates that fully decentralized, CRDT-based data management is feasible at scale; however, this design also exposes inherent trade-offs. Updates must traverse multiple hops before reaching all replicas, and convergence time is fundamentally bounded by the synchronization interval and overlay diameter. While this behaviour is acceptable for swarm-like and infrastructure-less environments, it makes Nimbus less suitable for applications that require low-latency global consistency or strict real-time guarantees.

The system's support for partial replication further contributes to temporary divergence between replicas. Nodes maintain only a subset of the global dataset and rely on

[51] https://codelab.fct.unl.pt/di/research/tardis/wp6/internal-tools/nimbus

opportunistic synchronization to receive updates, which naturally leads to greater staleness. While this design significantly reduces storage and communication overhead, it introduces the challenge of selecting appropriate replica sets for each information unit. Poor replica placement can increase the number of propagation paths and delay convergence, especially under churn or skewed access patterns. In addition, Nimbus's reliance on CRDTs introduces metadata and communication overhead, even when using delta-based optimizations.

Finally, our experience highlights several practical lessons from building and evaluating Nimbus. CRDTs successfully eliminate conflicts and prevent data loss, but they shift complexity toward efficient metadata management and propagation.

This work has shown that while solutions with weaker consistency semantics, such as Nimbus, provide behaviour well-suited to swarm-like and highly dynamic applications, careful consideration is required when selecting the appropriate data management approach for a given development context, operating conditions, and deployment environment. The lessons learned from the design and evaluation of Nimbus have directly informed the work carried out in WP6 and opened up several promising directions for future research.

4.6 Decentralised Application-Level Streaming

The contributions reported in this section address the problem of large-scale, decentralised dissemination of continuous data streams under real-time constraints. While previous sections focused on runtime support, execution substrates, and membership management for decentralised systems, effective support for swarm-based applications also requires mechanisms that enable timely and scalable one-to-many data distribution without reliance on centralised infrastructure.

Within WP6, this contribution is primarily associated with Task 6.1, as it builds on decentralised communication and coordination mechanisms, and with Task 6.3, given its direct relevance to efficient data dissemination and observability in large-scale swarms. The work is motivated by application scenarios where large numbers of nodes must receive continuous streams of information, such as situational awareness feeds, telemetry dissemination, or real-time coordination data, while operating under heterogeneous network conditions and without stable central coordination points.

The contribution explores application-level streaming as a decentralised service implemented over peer-to-peer overlays, with an explicit focus on scalability, robustness, and latency efficiency. Rather than proposing a single monolithic solution, the work systematically analyses design alternatives and evaluates a new decentralised streaming protocol against representative state-of-the-art approaches.

4.6.1 Context and Motivation

Contemporary large-scale streaming services rely almost exclusively on centralised infrastructures, typically implemented using globally distributed Content Delivery Networks. While these architectures provide strong performance guarantees, they require substantial infrastructure investment and introduce central points of control and

failure. In decentralised and edge-centric environments, such as those targeted by TaRDIS, these assumptions do not hold.

Application-level streaming over peer-to-peer overlays has long been proposed as an alternative, enabling the distribution of streaming content by leveraging the aggregate resources of participating nodes. However, existing decentralised streaming systems often face fundamental challenges related to scalability, heterogeneity of node capabilities, resilience to failures, and control overhead. Many solutions also implicitly assume relatively stable connectivity and homogeneous participants, assumptions that are increasingly violated in swarm-based and edge-centric deployments.

The motivation for this contribution is to revisit application-level streaming from a decentralised perspective, analysing existing approaches and proposing a design that explicitly accounts for large-scale deployments, heterogeneous node capabilities, and dynamic conditions. The goal is not to replace centralised streaming infrastructures, but to explore decentralised alternatives that are better aligned with the operational constraints and resilience requirements of TaRDIS use cases.

4.5.2 Objective and Design Goals

The primary objective of this contribution is to design and evaluate a decentralised application-level streaming protocol that disseminates real-time data streams to large numbers of receivers with low latency and controlled overhead. The design aims to support multiple concurrent streams, accommodate heterogeneous node capabilities, and remain robust under failures and churn.

Key design goals include scalability to thousands of nodes, avoidance of central coordination points, and efficient utilisation of available upload bandwidth across participants. The protocol is designed to balance dissemination latency against control overhead, recognising that aggressive optimisation of one dimension often degrades the other. Support for dynamic adaptation to changing conditions is treated as a first-class requirement, rather than an afterthought.

4.6.3 Technical Description

The decentralised application-level streaming solution developed within WP6 is designed to disseminate continuous data streams from one or more sources to large numbers of receivers without relying on centralised infrastructure. The protocol operates entirely at the application level and is structured around a peer-to-peer overlay that supports dynamic membership, heterogeneous node capabilities, and real-time dissemination constraints.

At a high level, the system combines two complementary overlay structures: a mesh overlay used for neighbour discovery and resilience, and a set of stream-specific dissemination trees used for efficient data forwarding. This hybrid design is motivated by the observation that purely tree-based approaches tend to minimise latency but are fragile under churn, while purely mesh-based approaches offer robustness at the cost of increased control overhead and delivery delay.

Each node maintains a bounded neighbour set, obtained through periodic peer sampling. Neighbour selection balances randomness and locality, ensuring sufficient connectivity while avoiding topological clustering. This mesh is continuously refreshed to tolerate

node joins, departures, and failures. Importantly, the mesh does not impose a strict structure on data dissemination; instead, it serves as a substrate over which dissemination trees are constructed and repaired.

For each active stream, a dissemination tree is incrementally established by selecting a parent node from the local neighbour set. Parent selection is guided by lightweight probing mechanisms that estimate peer responsiveness and communication latency. Nodes periodically reassess their parent choice, allowing dissemination paths to adapt dynamically as network conditions evolve. This process is fully decentralised and does not rely on global coordination or synchronisation.

Stream data is forwarded along the dissemination tree, with each node relaying received data to its children. Control messages are exchanged to maintain parent–child relationships and to detect failures. When a parent becomes unresponsive, a node locally selects an alternative parent from its neighbour set, triggering a local tree repair. This mechanism allows the system to recover from failures and churn without reconstructing the dissemination structure globally.

The protocol explicitly accounts for heterogeneous node capabilities. Nodes with higher available upload capacity naturally assume greater forwarding responsibility, as they are more likely to be selected as parents by multiple children. Conversely, constrained nodes tend to occupy leaf positions in the dissemination tree. This adaptive load distribution emerges from local decisions rather than from explicit role assignment or central scheduling.

Support for multiple concurrent streams is achieved by maintaining stream-specific dissemination state while sharing the underlying mesh overlay and peer sampling mechanisms. This design avoids redundant control overhead and enables efficient scaling as the number of streams increases.

Overall, the technical design prioritises low dissemination latency, robustness under churn, and scalability to large numbers of participants, while maintaining bounded control overhead and avoiding assumptions of stable connectivity or homogeneous nodes.

4.6.4 Relationship with Other WP6 Components

This contribution builds directly on the decentralised communication abstractions provided by the Babel ecosystem described in Section 4.1. The streaming protocol is implemented as a composition of decentralised protocols, benefiting from Babel's event-driven execution model and protocol modularity.

The overlay maintenance and dissemination mechanisms also relate closely to the membership management techniques discussed in Section 4.3, as effective streaming depends on the timely discovery of peers and accurate liveness information. Furthermore, the dissemination patterns explored here inform the design of telemetry aggregation mechanisms reported elsewhere in WP6, as both rely on efficient one-to-many and many-to-one communication patterns.

4.6.5 Advances Beyond the State-of-the-Art

Decentralised application-level streaming has been studied extensively, with a variety of systems proposed to address the limitations of centralised streaming infrastructures. Early approaches such as SplitStream leveraged structured overlays to construct multiple multicast trees, achieving good load distribution but exhibiting sensitivity to churn and overlay maintenance costs.[52] Mesh-based systems such as CoolStreaming and DONet emphasised robustness by exchanging buffer maps among peers, but incurred significant control overhead and often suffered from increased delivery latency under scale.[53]

More recent systems, including PPLive-style architectures and hybrid mesh–tree approaches, attempted to balance robustness and efficiency by combining gossip-based neighbour management with structured dissemination paths.[54] However, many of these systems assume relatively stable connectivity, homogeneous node capabilities, or rely on aggressive buffering strategies that are poorly suited to real-time dissemination with tight latency constraints.

The contribution reported here advances the state of the art by systematically combining mesh resilience with tree-based efficiency in a manner that is explicitly designed for large-scale, heterogeneous deployments. Unlike classical tree-based systems, the dissemination structure is not statically defined or centrally optimised; instead, it is continuously adapted through local decisions based on lightweight probing and failure detection. This enables rapid recovery from failures and churn while preserving low-latency dissemination paths.

In contrast to mesh-centric systems that rely on extensive state exchange (e.g., buffer maps) to ensure delivery, the proposed design limits control traffic by maintaining only minimal per-stream state and by sharing overlay maintenance mechanisms across streams. This results in lower and more predictable control overhead, particularly as system size increases.

Crucially, the work goes beyond analytical comparison by providing a comprehensive experimental evaluation against representative state-of-the-art systems, using delivery latency, control overhead, and robustness as primary metrics. The evaluation demonstrates that the proposed protocol achieves lower delivery latency than existing decentralised solutions while maintaining comparable control overhead, particularly in large-scale configurations involving thousands of nodes.

By explicitly addressing scalability, heterogeneity, and dynamic conditions within a unified design and evaluation framework, this contribution fills a gap between earlier decentralised streaming systems and the requirements of contemporary swarm-oriented and edge-centric applications targeted by TaRDIS.

---

[52] Castro, M., Druschel, P., Kermarrec, A.-M., and Rowstron, A., SplitStream: High-bandwidth multicast in cooperative environments, Proceedings of the 19th ACM Symposium on Operating Systems Principles, Bolton Landing, NY, USA, 2003, pp. 298–313

[53] Zhang, X., Liu, J., Li, B., and Yum, T.-S. P., CoolStreaming/DONet: A data-driven overlay network for peer-to-peer live media streaming, Proceedings of IEEE INFOCOM, Miami, FL, USA, 2005, pp. 2102–2111

[54] Hei, X., Liang, C., Liang, J., Liu, Y., and Ross, K. W., A measurement study of a large-scale P2P IPTV system, IEEE Transactions on Multimedia, vol. 9, no. 8, pp. 1672–1687, 2007

4.6.6 Experimental Validation and Evaluation

The experimental evaluation was designed to assess the scalability, efficiency, and robustness of the proposed streaming protocol under controlled yet realistic conditions. Experiments were conducted using a combination of simulation and real-world distributed deployments, enabling systematic exploration of large-scale scenarios.

Nodes were deployed as independent processes in a controlled cluster environment, with network conditions configured to emulate realistic latency and bandwidth constraints. The evaluation considers configurations ranging from hundreds to several thousand nodes; while results are reported across multiple scales, particular emphasis is placed on the largest configurations evaluated, as these best reflect the intended deployment scenarios.

Three classes of scenarios were considered: stable operation, crash failures, and churn. In stable scenarios, the focus is on steady-state performance and efficiency. Crash scenarios introduce sudden failures of a subset of nodes, while churn scenarios model continuous node arrivals and departures.

The primary metrics analysed include delivery rate, delivery latency, and upload load distribution. Delivery rate measures the fraction of stream messages successfully delivered to receivers. Delivery latency captures the end-to-end delay from the source to the receivers. Upload load distribution quantifies how forwarding responsibility is distributed across nodes.

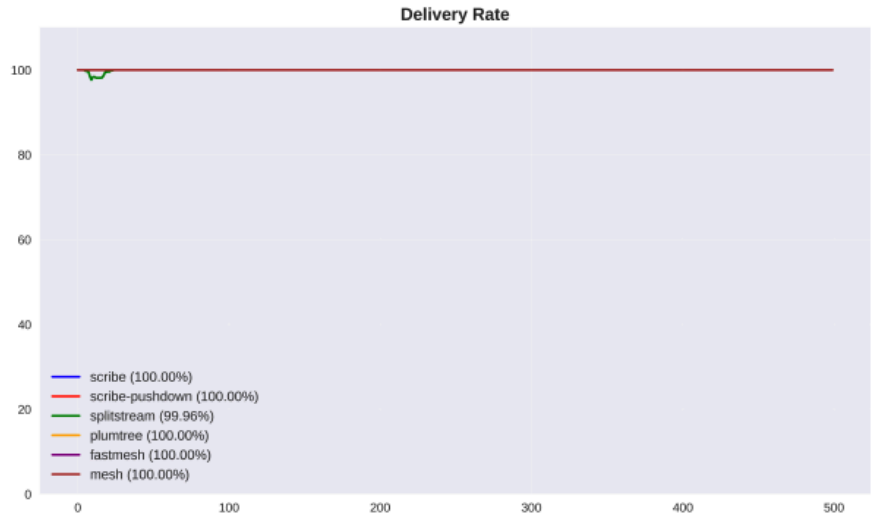

Figure 22: *Delivery Rate Evolution.*

Figure 22 shows the evolution of the delivery rate over time under stable conditions for the largest evaluated configuration. The results indicate that the protocol achieves near-complete delivery once the dissemination structure stabilises, demonstrating scalability to thousands of nodes.

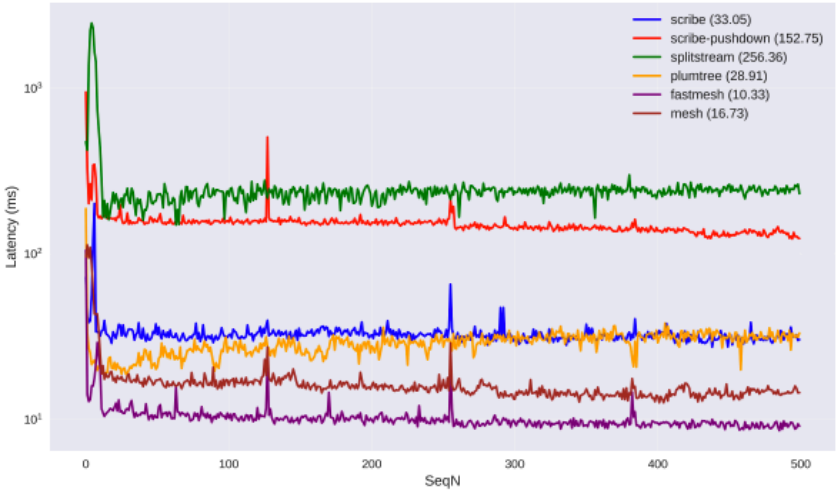

Figure 23: *Delivery Latency Evolution.*

Figure 23 reports delivery latency under stable conditions. The observed latency remains low and stable as the system scales, indicating that the tree-based dissemination paths effectively limit propagation delay even in large deployments.

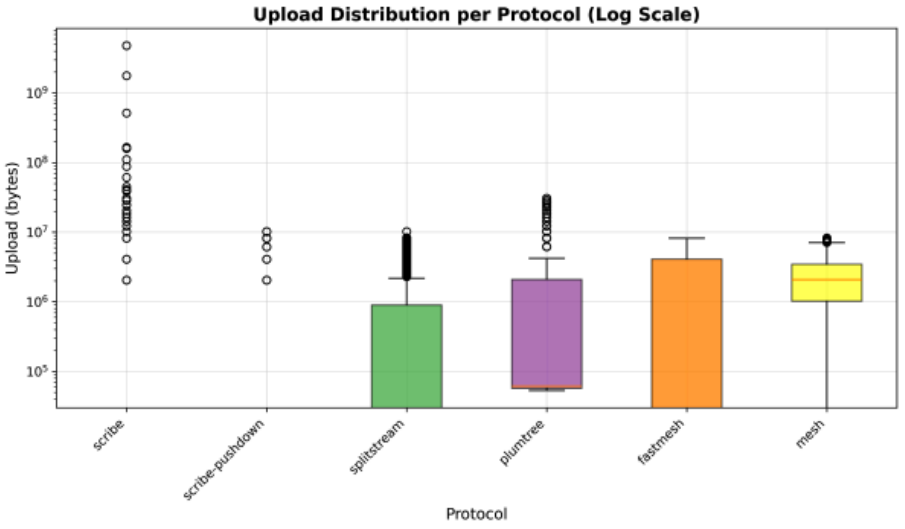

Figure 24: *Delivery Latency Evolution.*

Figure 24 illustrates the distribution of upload load across participating nodes. The results show that forwarding responsibility is spread across multiple nodes rather than concentrated on a small subset, confirming that the protocol avoids central bottlenecks and exploits aggregate upload capacity.

Crash and churn scenarios further demonstrate the protocol's robustness. While failures and churn temporarily reduce delivery rate and increase latency, the system recovers without global coordination, re-establishing effective dissemination paths through local adaptation mechanisms.

4.6.7 Limitations and Lessons Learned

The evaluation highlights several limitations inherent to decentralised application-level streaming. First, while scalability is demonstrated up to several thousand nodes, control

overhead grows with system size, imposing practical limits on extremely large deployments. Second, recovery from failures and churn introduces transient performance degradation, which may be problematic for applications with strict real-time requirements.

The results also underscore the importance of balancing mesh maintenance and tree optimisation. Overly aggressive optimisation can reduce robustness, while insufficient optimisation increases latency and overhead. These trade-offs must be carefully managed depending on application requirements.

Overall, the contribution demonstrates that decentralised application-level streaming is feasible and effective at scale, provided that protocol design explicitly accounts for heterogeneity, failures, and dynamic conditions. The lessons learned from this work informed subsequent WP6 efforts on dissemination, aggregation, and observability mechanisms.

4.7 Telemetry, Metrics, and Decentralised Monitoring

The contributions reported in this section address a fundamental yet often underexplored aspect of decentralised and swarm-based systems: the ability to observe, analyse, and reason about system behaviour at runtime through metrics and telemetry. While decentralised communication, coordination, and data management mechanisms enable such systems to operate without central control, the absence of adequate observability severely limits developers' and operators' ability to understand performance, diagnose failures, and evaluate design trade-offs.

Within WP6, this contribution is primarily associated with Task 6.3, complementing dissemination and aggregation mechanisms by focusing on the capture, aggregation, and presentation of runtime metrics in decentralised environments. It is also closely related to Task 6.1, as effective observability must integrate seamlessly with the communication and execution models provided by the underlying runtime frameworks. The work reported here aims to provide systematic support for metrics collection and real-time monitoring in decentralised systems, without reintroducing centralised bottlenecks or imposing intrusive instrumentation requirements on protocol developers.

4.7.1 Context and Motivation

Monitoring and performance assessment are well-established concerns in distributed systems engineering. In practice, developers rely heavily on metrics to evaluate correctness, performance, and resource utilisation during both development and operation. However, most existing monitoring solutions are designed for centralised or hierarchically managed infrastructures, such as data centres or cloud environments, where stable connectivity and well-defined administrative boundaries can be assumed.

In decentralised systems, and particularly in swarm-based and edge-centric deployments, these assumptions no longer hold. Nodes may join and leave dynamically, connectivity may be intermittent, and no single component can be assumed to have a global view of the system. As a result, widely adopted monitoring solutions often become impractical or introduce centralised points of failure and scalability bottlenecks.

The motivation for this contribution is to address this gap by designing a metrics collection and monitoring solution that is compatible with decentralised execution models.

The goal is to support both system-level metrics, such as hardware resource utilisation, and application or protocol-specific metrics, while preserving decentralisation principles and keeping instrumentation overhead low.

4.7.2 Objective and Design Goals

The primary objective of this contribution is to provide developers of decentralised protocols and applications with a uniform and extensible mechanism for collecting and analysing runtime metrics. The system is designed to support a wide range of metrics, including CPU usage, memory consumption, network I/O, and user-defined protocol metrics, without requiring substantial modifications to existing code.

A key design goal is non-intrusiveness. Instrumentation should require minimal effort from developers and should not significantly increase code complexity. At the same time, the system must introduce negligible runtime overhead, ensuring that collected metrics do not distort the behaviour being observed.

Another central objective is decentralised aggregation. Rather than relying on a single central monitoring component, the system supports multiple aggregation strategies that distribute the load of metrics collection and processing across participating nodes. This design aligns with the decentralised nature of the systems targeted by TaRDIS and avoids reintroducing central coordination points.

4.7.3 Technical Description

The proposed metrics collection system is structured as a modular architecture integrated with the Babel framework. At its core, the system provides a library that protocol developers can use to define and export metrics through a uniform interface. Metrics are treated as first-class entities, with explicit support for different metric types, including counters, gauges, and sampled values.

Each node runs a local metrics manager that collects metrics from the execution environment and instrumented protocols. System-level metrics are obtained by interfacing with the underlying operating system or container runtime, leveraging standard facilities exposed by the execution environment. Protocol-level metrics are explicitly defined by developers and reported through the same interface, enabling consistent handling and aggregation.

Collected metrics are periodically exported to one or more monitoring components. The system supports multiple exporter formats, allowing metrics to be encoded in different representations depending on the intended aggregation or visualisation pipeline. This flexibility enables integration with existing monitoring tools while maintaining control over how metrics are disseminated.

Aggregation is performed by specialised monitoring protocols that operate within the decentralised system itself. Several aggregation strategies are supported, ranging from simple local monitoring to hierarchical and clustered aggregation schemes. In more advanced configurations, designated nodes act as aggregation points, collecting metrics from subsets of the system and forwarding aggregated results further up the hierarchy. Importantly, these roles are defined at the protocol level and can be adapted dynamically, rather than being fixed infrastructure components.

The monitoring stack is completed by a metrics intake application and a visualisation layer. Aggregated metrics are stored in a time-series database and visualised using standard dashboard tools, enabling developers to inspect system behaviour in real time or retrospectively. While the visualisation layer may be deployed centrally for convenience, the collection and aggregation of metrics do not rely on centralised control.

4.7.4 Relationship with Other WP6 Components

This contribution integrates closely with the Babel ecosystem described in Section 4.1, extending it with native support for observability. By embedding metric collection into the runtime framework, the system ensures that any protocol developed using Babel can be instrumented consistently.

The aggregation mechanisms explored here also relate directly to the dissemination and aggregation patterns discussed in Sections 4.6 and 4.8. In particular, the design of decentralised monitors and aggregation managers informed later work on telemetry aggregation, where similar challenges arise regarding scalability and fault tolerance.

4.7.5 Advances Beyond the State-of-the-Art

Existing monitoring systems, such as SNMP, Nagios, Prometheus, and Graphite, provide robust solutions for centralised or hierarchically managed infrastructures, but they rely on assumptions that do not hold in decentralised environments. These systems typically assume stable endpoints, central collectors, or well-defined scraping hierarchies, making them ill-suited for systems with high churn and no central authority.

The contribution reported here advances the state of the art by embedding metrics collection directly into a decentralised execution framework. Rather than treating monitoring as an external service, metrics become an integral part of protocol execution, enabling consistent instrumentation across diverse protocols and applications.

Furthermore, the system explicitly supports decentralised aggregation strategies, allowing metrics to be collected and processed without requiring a single global collector. This contrasts with federation-based approaches, which still rely on stable intermediate collectors and are difficult to apply in highly dynamic systems.

By focusing on ease of instrumentation, low overhead, and decentralised operation, this work bridges the gap between traditional monitoring solutions and the needs of decentralised and swarm-based systems.

4.7.6 Experimental Validation and Evaluation

The experimental evaluation of this contribution focused on assessing both the usability and performance impact of the proposed metrics collection system. Experiments were conducted using distributed deployments of Babel-based protocols in controlled environments that enabled systematic variation of configuration parameters.

The evaluation considered multiple deployment scenarios, including a baseline configuration without instrumentation, a configuration with local monitoring, and configurations with decentralised aggregation using clustered monitors. Experiments were conducted with deployments ranging from small numbers of nodes to larger-scale

Funded by the European Union

configurations; particular emphasis is placed on the largest evaluated configurations, as these best reflect realistic decentralised deployments.

Metrics analysed include CPU usage, memory consumption, and network overhead introduced by the monitoring system. The results we obtained show that the overhead introduced by instrumentation is minimal compared to the baseline, with only marginal increases in CPU and memory usage. More advanced aggregation configurations introduce additional overhead at aggregation points, but this overhead remains bounded and predictable.

Importantly, the evaluation demonstrated that decentralised aggregation effectively distributes monitoring load, avoiding the bottlenecks observed in centralised approaches. The results confirm that the system can provide real-time observability without compromising the scalability or decentralisation of the underlying system.

4.7.7 Limitations and Lessons Learned

The evaluation highlighted several limitations inherent to decentralised monitoring. While decentralised aggregation improves scalability, it introduces additional complexity in terms of configuration and protocol design. Selecting appropriate aggregation strategies requires careful consideration of system size, churn, and desired observability granularity.

Additionally, while the system supports integration with existing visualisation tools, the reliance on external storage and visualisation components may still introduce practical deployment constraints in highly constrained environments.

Despite these limitations, the contribution demonstrates that decentralised metrics collection and real-time monitoring are feasible and practical when integrated directly into the execution framework. The lessons learned from this work informed subsequent WP6 efforts on telemetry aggregation and system-level observability.

4.8 Decentralised Telemetry Aggregation

The contributions presented in this section aim to address the problem of collecting, aggregating, and disseminating metrics in decentralized systems, particularly those operating at the edge or within swarm-based environments. Global aggregates should cover a wide range of metrics, ranging from hardware and system utilization (e.g., CPU usage, memory consumption, network I/O) to application-specific metrics generated by services running within these environments. The overall process must impose minimal overhead on participating nodes, avoid relying on any privileged nodes with superior capabilities or availability, and remain operational in dynamic conditions characterized by node churn, link failures, and changing network topology.

Within WP6, the contribution related to fault-tolerant and timely aggregation of system- and application-level metrics in swarm systems is primarily addressed in Task 6.3. It is also strongly connected to Task 6.1, which focuses on enabling the collection of user-defined protocol metrics without reintroducing centralized bottlenecks or requiring intrusive instrumentation from protocol developers, alongside the additional contributions described in other sections.

4.8.1 Context and Motivation

Collecting metrics in decentralized systems presents several challenges, including dynamic node membership, intermittent connectivity, and limited computational and networking resources. In such environments, no single component is responsible for maintaining a global view of the system, which significantly complicates monitoring and coordination. As a result, widely adopted monitoring solutions designed for structured cloud infrastructures often become impractical because they introduce centralized points of failure and scalability bottlenecks. Furthermore, systems operating in remote or intermittently connected environments cannot rely on these established monitoring approaches.

In swarm-based and edge-centric deployments, individual participants typically possess only local knowledge, whereas global information is often required to support system-wide decision-making. For example, edge environments comprising numerous resource-constrained devices require awareness of overall resource availability to enable effective scheduling, load balancing, or the prioritization of critical workloads under overload conditions. Additionally, applications executing in such environments may generate domain-specific or protocol-level metrics that must be collected and aggregated alongside standard system-level measurements.

Consequently, the motivation for this contribution is to address the limitations of existing monitoring approaches in decentralized environments by enabling efficient collection, aggregation, and dissemination of both system-level and application-level metrics. The resulting global aggregates can be used by other components of the toolbox for visualization, system optimization, and automated decision-making. At the same time, the proposed approach aims to preserve robustness under dynamic network conditions by continuously adapting the underlying structures used for metric collection, propagation, and aggregation, thereby maintaining operation in the presence of node churn, failures, and topology changes.

4.8.2 Objective and Design Goals

The primary objective of this contribution is to provide a decentralized protocol for computing global metric aggregates in large-scale environments with unreliable, dynamic network conditions. The protocol is intended to support the aggregation of both system- and container-level metrics (e.g., CPU usage, memory consumption, and network I/O) and user-defined protocol- or application-level metrics, enabling nodes to obtain a consistent global view of the system state.

To achieve efficient operation at scale, the protocol adopts a tree-based aggregation model, chosen for its significantly lower communication overhead compared to gossip-based aggregation approaches. In this model, each node periodically aggregates its locally collected metrics together with aggregated values received from its children and forwards the result to its parent. The root node computes the global aggregate and disseminates the result back down the tree, ensuring that all nodes obtain the most recent system-wide metrics. The protocol is designed to support any decomposable aggregation function and performs efficiently under stable network conditions.

To avoid introducing a single point of failure, the root node should not be statically assigned. Instead, the protocol incorporates mechanisms to dynamically select and replace root nodes when failures or topology changes occur, allowing the aggregation process to continue operating despite node or link failures. This dynamic adaptation is essential for maintaining robustness in swarm-based and edge-centric environments where node availability and connectivity may change frequently.

Node communication is limited to a small subset of peers provided by an underlying peer sampling service, thereby ensuring scalability and maintaining connectivity in highly dynamic networks. The system is designed to operate under realistic network conditions, including message loss, reordering, variable latency, and dynamic node membership, where nodes may join or leave at any time.

4.8.3 Technical Description

Aggregation and dissemination proceed in periodic rounds. In each round, every non-root node sends a local aggregation message to its parent, containing the aggregate of its own value and the most recent aggregates received from its children. The root node periodically computes the global aggregate and sends it to its children using global aggregation messages. Each non-root node forwards the most recent global aggregate it knows to its own children, ensuring dissemination throughout the tree. To avoid propagating invalid data, nodes forward global aggregates only once a valid value has been observed. This push-based dissemination ensures low latency and good scalability in stable conditions.

To maintain robustness in the presence of failures and changing network conditions, the protocol dynamically adapts the tree structure. The protocol builds on ideas from epidemic broadcast trees, in particular, the separation of neighbours into eager and lazy peers. Eager peers form the active tree and participate in forwarding aggregation messages, while lazy peers act as backup links that are not used for regular forwarding but are monitored for performance. Each node observes how quickly global aggregates arrive via its current parent compared to its lazy peers. If a lazy peer consistently delivers updates faster than the current parent, the node may switch to that peer as its new parent, provided that a configurable threshold is exceeded. This mechanism allows the tree to adapt to changing link quality while limiting excessive instability that could reduce aggregation accuracy.

Tree consistency is maintained through simple local rules that reconcile how two peers perceive their relationship. Valid configurations include parent–child and lazy–lazy relationships. Any inconsistent combinations are resolved automatically based on message exchanges and node identifiers, preventing persistent two-node cycles. Longer cycles are broken over time because lag measurements eventually trigger parent changes. When new peers appear, they are initially treated as children. Failed peers are detected based on missing messages over a bounded number of rounds and are removed from the tree. If no tree is active, nodes periodically exchange lightweight ping messages to maintain failure detection.

Root election is decentralized and probabilistic. When a node detects that no global aggregate has been received for a sufficiently long time, it assumes that the current root has failed and discards the associated tree state. Nodes without an active tree

periodically attempt to elect themselves as root with a probability inversely proportional to an estimated total number of nodes. This ensures that, with high probability, at least one new root emerges quickly after a failure, while limiting the number of simultaneous roots. Temporary coexistence of multiple roots is tolerated: nodes always prefer the tree whose root has the highest identifier, and lower-identifier roots eventually stop propagating, causing their trees to disappear. The node count estimate is derived from the aggregation itself and is updated only after the root has completed a sufficient number of rounds, reducing the risk of severe underestimation.

4.8.4 Relationship with Other WP6 Components

The described decentralized aggregation metrics protocol can accept metrics from other parts of the toolbox. As mentioned in Section 4.8. Task 6.1 focuses on the collection of user-defined protocol metrics without reintroducing centralised bottlenecks or imposing intrusive instrumentation requirements on protocol developers, amongst the other contributions shown in previous sections. Such data can serve as input to the aggregation process, enabling nodes to obtain a more informative and consistent global view of the system.

4.8.5 Advances Beyond the State-of-the-Art

The proposed approach improves upon traditional centralized aggregation solutions by eliminating reliance on dedicated monitoring infrastructure, which often introduces scalability limitations and single points of failure. By distributing metric collection and aggregation across participating nodes, the protocol enables operation in dynamic, resource-constrained, and intermittently connected swarm and edge environments. Unlike hierarchical aggregation methods that rely on statically assigned roots or fixed overlay structures, the proposed protocol supports dynamic root selection and adaptive aggregation structures, allowing aggregation to continue despite node failures, churn, or topology changes.

In contrast to probabilistic or sketch-based aggregation techniques that yield approximate results, the proposed approach enables deterministic aggregation while maintaining low communication overhead via hierarchical aggregation. Additionally, the protocol is not limited to predefined aggregation functions and supports any decomposable aggregation operation, allowing integration of both system-level and application-specific metrics. This flexibility makes the approach suitable for a wide range of decentralized monitoring and decision-making scenarios.

4.8.6 Experimental Validation and Evaluation

The proposed protocol was evaluated through extensive simulations designed to reflect realistic swarm and edge deployment conditions. The simulation scenarios included dynamically changing metric values, node and link failures affecting varying percentages of participants, root node failures, and temporary node disconnections. Additionally, network behaviour such as message latency and packet loss was simulated to emulate unreliable communication environments.

The evaluation results confirm that the aggregation process remains successful across a wide range of tested scenarios. The protocol consistently produces reliable aggregation results under stable conditions and can recover from node failures, link disruptions, and

root node failures. These findings indicate that the adaptive aggregation mechanisms allow the system to maintain operational continuity and preserve aggregation correctness despite dynamic and unreliable network conditions.

4.8.7 Limitations and Lessons Learned

Despite its advantages, tree-based aggregation remains inherently sensitive to structural disruptions, as failures of parent nodes or critical links may temporarily interrupt aggregation flows. While the proposed protocol mitigates this fragility through adaptive behaviour, including dynamic restructuring and root replacement mechanisms, short periods of aggregation unavailability may still occur during topology repair or root re-election.

Nevertheless, the results demonstrate that it is feasible to compute global aggregates in decentralized swarm environments with relatively low additional communication overhead. The obtained aggregates can serve multiple purposes, including system monitoring, optimization, and decision support, confirming that hierarchical aggregation remains a practical and efficient approach when combined with adaptive mechanisms designed for dynamic and unreliable networks.

4.9 Network Emulation and Experimental Infrastructure

The contributions reported in this section address a transversal and enabling concern for the validation of decentralised systems: the ability to experimentally evaluate protocol behaviour under realistic, large-scale, and dynamically changing network conditions. While earlier sections focused on the design and implementation of decentralised protocols and runtime mechanisms, their validation necessarily depends on experimental environments that can reproduce complex network behaviours at scale.

Within WP6, this contribution supports all technical tasks by providing an experimental substrate that enables controlled and repeatable evaluation of decentralised mechanisms under adverse and heterogeneous network conditions. In particular, it plays a central role in enabling experimental validation at scales that are otherwise impractical to achieve with physical testbeds, aligning directly with the TaRDIS objective of demonstrating feasibility and robustness for systems involving up to several thousand interacting entities.

4.9.1 Context and Motivation

Experimental evaluation is a cornerstone of distributed systems research. However, as decentralised systems grow in scale and complexity, reproducing realistic network conditions becomes increasingly challenging. Physical testbeds, while accurate, are limited by cost, availability, and scalability. Cloud infrastructures offer elasticity but provide limited control over low-level network properties and introduce significant operational costs when scaling to thousands of nodes.

Simulation-based approaches enable large-scale experimentation but rely on abstract models and simulated time, often diverging from real execution behaviour. Emulation provides an attractive middle ground, allowing real applications to execute in real time while interacting with an emulated network that reproduces relevant characteristics such as latency, bandwidth constraints, jitter, and packet loss.

The motivation for this contribution is to address the limitations of existing network emulation tools when applied to large-scale decentralised systems. In particular, existing solutions often require static configuration of network properties prior to experiment execution, limiting their usefulness for studying adaptive protocols whose behaviour depends on dynamic network conditions. Furthermore, many tools struggle to scale beyond a few hundred nodes without incurring prohibitive overhead.

4.9.2 Objective and Design Goals

The primary objective of this contribution is to provide a generic and scalable network emulation framework that supports the experimental evaluation of decentralised systems with thousands of nodes. The framework is designed to enable dynamic modification of network properties at runtime, enabling experiments that respond to system behaviour or external stimuli.

Key design goals include scalability, flexibility, and realism. Scalability is required to support experiments involving several thousand nodes, in line with TaRDIS validation objectives. Flexibility is achieved by exposing a programmable interface that allows network topologies and properties to be modified dynamically. Realism is ensured by executing unmodified application code in real time, interacting with an emulated network that closely approximates real-world behaviour.

4.9.3 Technical Description

The proposed solution is a network emulation framework designed around containerisation and kernel-level packet processing. Applications are deployed as containers, enabling large numbers of networked entities to be instantiated on a limited set of physical machines while maintaining strong isolation between nodes. Each container is treated as an independent network endpoint, executing unmodified application code.

Network emulation is achieved by intercepting and manipulating network traffic at the kernel level. The framework leverages high-performance packet-processing mechanisms to model network properties such as latency, bandwidth limitations, jitter, and packet loss. By operating at low levels of the network stack, the emulation remains transparent to the applications under test.

A central architectural element is the separation between application execution and network control. Network behaviour is managed through a dedicated control plane that exposes an API for configuring network components, connections, and properties. This API allows the experimenter to define network topologies, inject faults, and modify link characteristics dynamically while the experiment is running.

The framework supports both local and distributed deployment modes. In local mode, all containers are deployed on a single host, enabling cost-effective experimentation at moderate scales. In distributed mode, containers are spread across multiple physical machines, allowing the framework to scale to several thousand nodes while preserving accurate network emulation semantics.

Dynamic reconfiguration is a distinguishing feature of the framework. Network properties can be altered during execution, enabling the emulation of scenarios such as link failures, congestion, topology changes, and intermittent connectivity. This capability is

particularly relevant for evaluating decentralised protocols designed to adapt to changing conditions, as it allows experiments to capture transient behaviours that static configurations cannot reproduce.

4.9.4 Relationship with Other WP6 Components

This contribution underpins the experimental validation of several WP6 results. The decentralised communication, dissemination, and aggregation mechanisms described in Sections 4.3, 4.5, and 4.7 rely on large-scale experimental evaluation to assess scalability and robustness. The network emulation framework provides the means to conduct such evaluations under controlled yet realistic conditions.

Furthermore, the ability to dynamically modify network conditions complements the observability mechanisms discussed in Section 4.6, enabling experiments that correlate protocol behaviour with changes in network state. As such, this contribution acts as a cross-cutting enabler for experimental work within WP6.

4.8.5 Advances Beyond the State-of-the-Art

Existing network emulation tools such as ModelNet, Mininet, and Kollaps have demonstrated the feasibility of emulating network behaviour for distributed systems. However, these tools typically require network configurations to be defined prior to experiment execution and offer limited support for dynamic reconfiguration at runtime. As a result, they are poorly suited for studying adaptive protocols whose behaviour evolves in response to changing network conditions.

The contribution reported here advances the state of the art by providing a generic emulation framework that combines scalability with dynamic control. By leveraging containerisation and kernel-level packet processing, the framework achieves performance comparable to existing solutions while enabling runtime modification of network properties through a programmable interface.

Unlike approaches that rely on simulation or hybrid simulation–emulation models, the framework executes real application code in real time, preserving interactions with the operating system and networking stack. This design choice ensures that observed behaviour closely matches that of real deployments, an essential requirement for validating decentralised systems intended for deployment in operational environments.

4.9.6 Experimental Validation and Evaluation

The experimental evaluation focuses on assessing the accuracy, scalability, and flexibility of the network emulation framework. Experiments were conducted using both synthetic benchmarks and representative distributed applications, executed within the emulated environment.

Evaluation scenarios include emulating latency, bandwidth constraints, jitter, and packet loss, with results compared against expected values and existing emulation tools. The framework accurately enforces configured network properties in both local and distributed deployments.

Scalability is evaluated by progressively increasing the number of emulated nodes. Experiments demonstrate that the framework can support deployments involving several

thousand containers, distributed across multiple physical machines, while maintaining stable operation and predictable overhead. This directly aligns with TaRDIS objectives of experimentally validating decentralised solutions at scales approaching 5000 nodes.

Dynamic behaviour is evaluated by modifying network properties during execution. Experiments show that changes in latency, bandwidth, and connectivity are promptly reflected in application-level behaviour, without requiring experiment restarts. This capability enables the study of transient phenomena such as network disruptions and recovery, which are critical for decentralised and swarm-based systems.

Overall, the evaluation confirms that the framework provides a practical and effective platform for large-scale experimental validation of decentralised systems, combining realism, scalability, and dynamic control.

4.9.7 Limitations and Lessons Learned

While the framework demonstrates strong scalability and flexibility, the evaluation highlights several limitations. Emulating very large topologies remains constrained by the available physical resources, particularly CPU and memory, and careful deployment planning is required to achieve maximum scale.

Additionally, while dynamic reconfiguration is supported, complex scenarios with frequent, large-scale topology changes may introduce non-negligible overhead. These trade-offs highlight the inherent tension between realism and scalability in network emulation.

Despite these limitations, the contribution demonstrates that large-scale, dynamic network emulation is feasible and effective for validating decentralised systems. The lessons learned from this work informed the experimental methodology adopted across WP6 and contributed directly to the experimental validation of TaRDIS solutions at significant high scales.

4.10 Secure and Privacy-Aware Decentralised support for Decentralized Markets

The contribution reported in this section explores the application of decentralised communication and coordination mechanisms to the design of secure and privacy-aware resource markets. While previous sections focused on foundational runtime, dissemination, observability, and aggregation mechanisms, this section demonstrates how such mechanisms can be composed into a concrete application-level system that addresses real-world coordination challenges under strict privacy and trust constraints.

Within WP6, this work serves a dual role. On the one hand, it provides a realistic validation context for the decentralised abstractions developed in Tasks 6.1 and 6.3, exercising them under demanding application requirements. On the other hand, it advances the state of the art in decentralised coordination by addressing security and privacy concerns that are intrinsic to resource trading scenarios, particularly in domains such as local energy markets. The contribution is explicitly aligned with TaRDIS objectives by evaluating decentralised coordination at scale and by grounding its design and validation in experimentally reproducible settings.

4.10.1 Context and Motivation

Decentralised resource markets constitute an important class of swarm-oriented applications targeted by TaRDIS, where autonomous nodes must coordinate economic exchanges without relying on continuously available central infrastructure. In such systems, decentralisation is not only a matter of scalability and fault tolerance, but also a fundamental requirement to reduce trust concentration, limit information exposure, and preserve user privacy.

Within WP6, this contribution complements the communication, runtime, and membership mechanisms described in earlier sections by demonstrating how decentralised coordination primitives can be composed into a concrete, privacy-aware application-level protocol. The focus is on local energy markets, where households, electric vehicles, and small producers exchange energy offers and bids directly. This domain is particularly demanding, as communication patterns are highly dynamic, participants are heterogeneous, and privacy violations may reveal sensitive behavioural information.

The work reported in this section addresses these challenges by designing, implementing, and experimentally evaluating decentralised market protocols that explicitly balance privacy, scalability, and operational feasibility. The contribution is aligned with TaRDIS objectives by validating decentralised coordination at scale and by grounding the design in realistic deployment assumptions derived from smart-grid environments.

4.10.2 Objective and Design Goals

The primary objective of this contribution is to enable decentralised resource trading while limiting the exposure of sensitive participant information and avoiding reliance on continuously trusted central entities. To this end, the system is designed around the following goals:

- **Decentralised coordination**: Resource offers and bids should be disseminated and matched without requiring a central broker to observe all interactions.
- **Privacy preservation**: Information about individual consumption, production, and bidding behaviour should not be globally visible, and linkability between actions should be minimised.
- **Scalability**: The system must operate with thousands of participants, reflecting realistic community-scale deployments and TaRDIS experimental targets.
- **Operational realism**: The design must tolerate heterogeneous devices, variable connectivity, and partial failures while remaining compatible with regulatory and settlement requirements.

To explore the design space, two complementary architectures are considered: an operator-assisted hierarchical solution and a fully peer-to-peer decentralised market. Their comparison provides insight into the trade-offs between privacy, complexity, and performance.

4.10.3 Technical Description

Architectural Overview

The proposed system is structured as a layered decentralised application built on top of peer-to-peer communication and membership services. At the lowest layer, nodes rely on an unstructured overlay network to maintain connectivity under churn. On top of this overlay, dissemination mechanisms are used to propagate market information, such as offers and bids, with locality bias.

At the application layer, each node acts as an autonomous market participant. Nodes periodically publish offers to sell or buy resources, annotated with price, quantity, and locality constraints. Matching decisions are performed in a decentralised manner, based on locally available information, without requiring global state.

Two architectural variants are implemented:

- **Hierarchical operator-assisted market**: Nodes are grouped under intermediary managers who aggregate information and facilitate coordination. This approach aligns with existing power-grid hierarchies and simplifies monitoring and settlement.
- **Pure peer-to-peer market**: All nodes are equal participants, exchanging market information directly and interacting with the operator only for certification and settlement purposes.

Privacy and Security Mechanisms

To address impersonation, replay, and repudiation attacks, all market messages are authenticated. The design employs group signature mechanisms that enable participants to verify that messages originate from legitimate members without revealing their identities to other peers. A trusted authority can reveal the signer in the event of disputes, which is necessary for regulatory compliance.

By avoiding global dissemination of fine-grained behavioural data and limiting message visibility to local neighbourhoods, the system reduces privacy leakage compared to centralised markets. The peer-to-peer variant further limits operator visibility, as the operator does not observe individual market interactions in real time.

Implementation Considerations

The system is implemented as a modular protocol stack, enabling the reuse of communication and membership services across both architectural variants. The market logic itself is intentionally simple, focusing on correctness and scalability rather than on sophisticated pricing heuristics. This choice reflects the goal of evaluating coordination mechanisms rather than market optimisation strategies.

An integration with an electric-grid simulator enables realistic workload generation, allowing nodes to produce and consume energy according to configurable profiles. This integration is essential for evaluating the system under realistic temporal and spatial demand patterns.

4.10.4 Relationship with Other WP6 Components

This contribution builds directly on the decentralised communication and membership mechanisms described in Sections 4.1 to 4.3, reusing overlay maintenance, dissemination, and failure-tolerance properties. It also benefits from the runtime abstractions that allow protocols to be composed and executed consistently across heterogeneous environments.

In turn, the market protocols provide a concrete application context that exercises WP6 mechanisms under realistic and demanding conditions. The results inform the design of observability and monitoring components (Section 4.7), as market performance metrics are derived from decentralised measurements rather than central logs.

4.10.5 Advances Beyond the State-of-the-Art

Existing decentralised energy market proposals often rely on either fully centralised brokers or blockchain-based infrastructures, both of which introduce limitations in scalability, latency, or privacy. In contrast, the contribution reported here demonstrates that decentralised, non-blockchain peer-to-peer markets can be implemented with strong privacy properties and practical performance.

The systematic comparison between hierarchical and pure peer-to-peer designs provides empirical evidence of the trade-offs involved, rather than relying solely on analytical arguments. In particular, the work shows that meaningful decentralisation can be achieved without sacrificing scalability, while still remaining compatible with regulatory constraints.

By grounding the evaluation in realistic workloads and by scaling experiments to thousands of nodes, this contribution bridges the gap between conceptual decentralised market designs and deployable systems suitable for TaRDIS use cases.

4.10.6 Experimental Validation and Evaluation

Experimental Setup

The evaluation is conducted through simulation-based experiments, using the integrated grid simulator to generate realistic production and consumption patterns. Network sizes range from small deployments to several thousand nodes, with particular emphasis on the largest configurations to assess scalability.

Metrics include message dissemination overhead, hop count, redundancy, deliverability, and market-level outcomes such as trade acceptance rates. Both architectural variants are evaluated under comparable conditions.

Summary of Results and Discussion

The results show that both architectures scale to large network sizes while maintaining high message deliverability. The peer-to-peer variant exhibits slightly higher control overhead due to fully decentralised dissemination but achieves stronger privacy properties by reducing operator visibility.

Trade acceptance rates remain stable as network size increases, indicating that locality-biased dissemination effectively limits unnecessary propagation. The experiments

also show that decentralised matching can be performed with predictable latency, even under high churn.

Overall, the evaluation confirms that decentralised resource markets are feasible at the scales targeted by TaRDIS, and that privacy-aware designs need not compromise performance.

4.10.7 Limitations and Lessons Learned

The evaluation highlights several limitations. First, the simplicity of the market heuristics limits economic optimality, which was intentionally left out of scope. Second, the reliance on simulation, while necessary for scale, cannot capture all real-world network effects.

From a design perspective, the comparison between architectures shows that full decentralisation increases protocol complexity, particularly in monitoring and debugging. Hierarchical designs offer operational advantages but at the cost of increased trust in intermediary nodes.

These observations inform future work after the conclusion of TaRDIS, particularly regarding adaptive hybrid architectures that combine decentralised coordination with selective operator assistance.

5. Cross-Cutting Integration and System-Level View

5.1 Final WP6 architecture

The final architecture emerging from the work carried out in WP6 reflects a layered and modular view of decentralised system support, in which a small set of well-defined runtime abstractions provides the foundation for higher-level coordination, data management, and observability mechanisms. Rather than converging on a monolithic platform, WP6 has deliberately evolved a composable architecture in which individual components can be selectively combined and specialised according to application requirements, deployment constraints, and target environments.

At the core of this architecture lies the execution and coordination substrate provided by the Babel ecosystem and its derivatives, including Micro-Babel for constrained and embedded devices. These runtimes establish a common execution model based on event-driven interaction, protocol composition, and dynamic reconfiguration, enabling decentralised components to discover peers, exchange messages, and adapt their behaviour at runtime. On top of this substrate, specialised mechanisms for membership management, replicated data handling, application-level streaming, and decentralised monitoring can be deployed as independent but interoperable building blocks.

A key architectural characteristic of WP6 is the explicit separation between mechanisms and policies. Core components expose generic primitives—such as peer sampling, message dissemination, state replication, or metric collection—while leaving policy decisions (e.g., overlay structure, dissemination strategies, consistency trade-offs, or reconfiguration triggers) to be defined by the application or higher-level tools. This separation has proven essential to support the wide range of scenarios addressed in TaRDIS, from relatively stable edge deployments to highly dynamic and intermittently connected environments.

Overall, the final WP6 architecture provides a coherent system-level view in which communication, coordination, data management, and observability are treated as first-class and interrelated concerns, while remaining sufficiently decoupled to support incremental adoption and evolution beyond the scope of the project.

5.2 Interoperability across WP6 components

Interoperability across WP6 components has been addressed as a fundamental design objective throughout the project, rather than as a post-hoc integration effort. This is reflected in the consistent use of shared abstractions, uniform interaction patterns, and explicit integration points across the different technical contributions reported in Section 4.

At runtime, interoperability is primarily achieved through the event-based execution model promoted by the Babel ecosystem. Components such as membership protocols, streaming mechanisms, telemetry collectors, and reconfiguration services interact via well-defined events and callbacks, enabling composition without tight coupling or mutual assumptions about internal state. This approach enables multiple components to coexist

within the same execution context, while preserving isolation and facilitating independent evolution.

Data-oriented components, including replicated storage and decentralised data management mechanisms, have been designed to integrate seamlessly with communication and coordination layers. In particular, data dissemination, consistency maintenance, and state synchronisation rely on the same peer discovery and messaging primitives provided by the underlying runtime, ensuring that data-related functionality remains aligned with the dynamic behaviour of the system as a whole.

Similarly, monitoring and telemetry mechanisms are not treated as external add-ons, but as integral parts of the runtime architecture. Metrics collection, aggregation, and exposure are closely integrated with execution and communication components, enabling fine-grained observability and supporting feedback loops for adaptive behaviour and reconfiguration.

This emphasis on interoperability has resulted in a WP6 toolbox in which individual contributions can be reused in isolation but also combined to form more comprehensive system configurations. The experience gained during integration activities confirms that this approach significantly reduces integration overhead and lowers the barrier for experimenting with alternative architectural compositions.

5.3 Relationship with other Technical Work Packages

WP6 maintains close relationships with other technical packages in the project by providing the fundamental mechanisms for communication, coordination, data management, and observability.

Beyond the generic tools and designs offered by WP6 in its final architecture, the team has developed various adaptors to interconnect tools from other work packages, providing straightforward options for combining components from the TaRDIS Toolbox. In the following subsections, we describe some of the results obtained through this integration effort.

5.3.1 WP5

5.3.1.1 PTB-FLA

Python Testbed for Federated Learning Algorithms (PTB-FLA) is a lightweight federated learning framework that provides abstractions for easy development of FL algorithms and applications.

In collaboration with WP5, WP6 developed an adaptor connecting PTB-FLA applications to the communication protocols implemented in the Babel Ecosystem. This adaptor allows PTB-FLA applications that would normally run on a single machine to be executed in a distributed manner by leveraging Babel's communication mechanisms and taking full advantage of the library's available protocols.

The source code for the adapter can be found in the WP6 Adapters repository[55].

[55] https://codelab.fct.unl.pt/di/research/tardis/wp6/babel-swarm/adapters/fauno

5.3.1.2 FAuNO

FAuNO is a federated AI network orchestrator that enhances the coordination and operation of distributed AI systems. Leveraging federated learning principles, FAuNO enables privacy-preserving collaboration across multiple nodes while maintaining high performance and scalability. WP6 developed a dedicated FAuNO adapter that interfaces with Babel through a well-defined API. This enables FAuNO nodes to communicate efficiently, exchange information related to training and decision-making and deploy federated AI workloads in a distributed and coordinated fashion.
The source code for the adapter can be found under the WP6 adapters[56].

5.4 Alignment with WP7 demonstrations

The alignment between WP6 developments and the demonstration and validation activities in WP7 has been a primary concern throughout the project. While WP6 focuses on the design and implementation of reusable runtime mechanisms, WP7 provides the context in which these mechanisms are exercised, combined with higher-level tools, and assessed in realistic scenarios.

Several WP6 components have been directly adopted in WP7 demonstrations, most notably the Babel ecosystem, which served as a coordination and communication substrate in multiple use cases. In these settings, WP6 mechanisms supported peer discovery, decentralised messaging, dynamic configuration, and interaction with learning and optimisation components developed in other work packages. Other WP6 contributions, such as decentralised monitoring and data management mechanisms, informed the design of demonstrators even when not fully deployed along critical execution paths.

Equally important, WP7 demonstrations provided concrete feedback that influenced the refinement and consolidation of WP6 components. Integration constraints, deployment challenges, and observed runtime behaviour in heterogeneous environments contributed to the identification of limitations, the clarification of design trade-offs, and the prioritisation of robustness and composability over feature completeness. In this sense, the interaction between WP6 and WP7 should be seen as bidirectional, with demonstrators both exploiting and shaping the WP6 toolbox.

It is also worth noting that not all WP6 components were intended to be uniformly integrated into all demonstrations. The selective adoption observed in WP7 reflects differences in architectural fit, maturity requirements, and prototype scope, rather than shortcomings of the underlying mechanisms. Nevertheless, the demonstrators collectively validate the relevance of the abstractions and design principles developed in WP6 and provide evidence of their applicability in diverse decentralised and data-driven systems.

[56] https://codelab.fct.unl.pt/di/research/tardis/wp6/demonstrations/gmv/adapters

6. Overview of the Main Results Produced by WP6

This section provides a short overview of the main results produced by WP6, as reported in this deliverable and in deliverables D6.1 and D6.2. For conciseness, we omit results that have since been discontinued (the reasons for this are discussed in D6.2). Each result is listed with a brief description, its primary WP6 task affiliation, the deliverable(s) in which it is reported, and its status at project conclusion. Detailed technical descriptions, evaluations, and limitations are provided in Section 4 of this deliverable and in the corresponding sections of D6.1 and D6.2.

Runtime and Execution Infrastructure

Babel-Swarm — Autonomic extension of Babel for large-scale swarm environments, adding self-configuration (pluggable discovery without preconfigured contacts), certificate-based identity management, SecureChannel abstraction for encrypted communication, and adaptive parameter management informed by decentralised network-size estimation. Primary task: T6.1. Reported in: D6.2, D6.3 (Section 4.1). Status: mature; validated at 5,000 nodes.

Babel-Android — Mobile variant of Babel-Swarm, encapsulating the protocol runtime within an Android foreground service to preserve active connections despite Android lifecycle management. Primary task: T6.1. Reported in: D6.2, D6.3 (Section 4.1). Status: mature; validated on seven Android device models for 48+ hours on battery.

Micro-Babel (µBabel) — Embedded variant of the Babel programming model, implemented in C on FreeRTOS for resource-constrained devices (ESP32, RP2040). Protocols map to FreeRTOS tasks with blocking queues; communication abstractions cover LoRa, Wi-Fi, and BLE; a Network Manager provides first-class connectivity awareness. Primary task: T6.1. Reported in: D6.2 (initial design), D6.3 (Section 4.2). Status: validated; integrated into the TaRDIS Messaging App demonstrator.

Babel API for Web Services — REST and WebSocket interface exposing Babel-Swarm runtime functionality to non-Java applications, enabling integration with TypeScript, Python, and JavaScript clients. Primary task: T6.1. Reported in: D6.2 (Section 4.1). Status: mature; used in the Messaging App web interface.

Decentralised Membership and Overlay Management

HyParView — Membership protocol with active/passive view separation, providing robust overlay connectivity under churn. Multiple variants developed: base, secure (certificate-authenticated joins and connections), autonomic (adaptive view sizes based on Random Tour estimates), and self-discovery (contact-free bootstrap via multicast/broadcast/DNS). Primary task: T6.1. Reported in: D6.1 (Section 4.3). Status: mature; validated at 5,000 nodes with 99.49% reliability.

Cyclon — Lightweight gossip-based peer sampling protocol providing continuously refreshed partial views through periodic random neighbour exchange. Primary task: T6.1. Reported in: D6.1. Status: mature.

X-BOT — Adaptive overlay optimisation protocol that reshapes the overlay topology based on application-specific criteria (e.g., latency, bandwidth) while preserving global connectivity. Primary task: T6.1. Reported in: D6.1. Status: mature.

Random Tour — Decentralised network-size estimation protocol based on random walks, providing input to autonomic management of protocol parameters. Primary task: T6.1/T6.3. Reported in: D6.2. Status: mature; integrated with Babel-Swarm autonomic controller.

Membership for Intermittent Networks — Membership variants addressing time-varying and partially connected topologies, motivated by satellite swarm and mobile scenarios. Primary task: T6.1. Reported in: D6.2, D6.3 (Section 4.3). Status: validated in emulation.

Communication and Dissemination

Eager Gossip Broadcast — Epidemic broadcast protocol with configurable fanout, providing probabilistic reliable delivery through redundant message forwarding. Secure variant adds message signing and certificate-authenticated channels. Primary task: T6.1. Reported in: D6.1. Status: mature; validated at 5,000 nodes.

Flood Broadcast — Deterministic broadcast forwarding all messages to all known neighbours, used as baseline in experimental comparisons. Primary task: T6.1. Reported in: D6.1. Status: mature.

One-Hop Broadcast — Lightweight broadcast primitive disseminating messages to immediate overlay neighbours only, for use cases requiring low overhead rather than full coverage. Primary task: T6.1. Reported in: D6.2. Status: mature.

Anti-Entropy Protocol — A periodic state reconciliation mechanism that detects and recovers missed messages after gossip delivery, providing convergence to 100% reliability. Secure variant adds encryption and signing. Primary task: T6.1. Reported in: D6.2. Status: mature; validated at smaller scales (excluded from 5,000-node experiment due to memory constraints).

Replicated Data Management

Nimbus — Fully decentralised CRDT-based key-value store with partial replication, access control policies on keyspaces and collections, and anti-entropy-based state reconciliation. Built directly on Babel-Swarm. Primary task: T6.2. Reported in: D6.2, D6.3 (Section 4.5). Status: validated through the TaRDIS Voting App (simple demo); partial replication and access control under active refinement.

CRDT Library — Shared library of conflict-free replicated data types (counters, registers, sets, maps) used by Nimbus and other storage components. Primary task: T6.2. Reported in: D6.2. Status: mature.

PotionDB — Transactional key-value store with CRDT-based conflict resolution, supporting geo-distributed partial replication with materialised views. Primary task: T6.2. Reported in: D6.2, D6.3. Status: validated through standalone evaluation. We note that we have not integrated PotionDB on any demonstrator or use case prototype, as the properties of PotionDB, while useful to build swarm applications, were not required in those contexts.

Arboreal — Hierarchical cloud-edge replication system supporting causal+ consistency with dynamic extension of replication towards edge locations. Primary task: T6.2. Reported in: D6.1, D6.2. Status: validated through emulation-based experiments. We

note that we have not integrated Arboreal on any demonstrator or use case prototype, as the properties of Arboreal, while useful to build swarm applications, were not required in those contexts.

External Adapters and Integration

Cassandra Adapter — Adapter exposing the TaRDIS-defined storage API over Apache Cassandra. Primary task: T6.2. Reported in: D6.1. Status: completed.

Hyperledger Fabric Adapter — Adapter exposing the TaRDIS-defined storage API over IBM Hyperledger Fabric. Primary task: T6.2. Reported in: D6.1. Status: completed.

C3 Adapter — Adapter for the C3 industrial IoT platform. Primary task: T6.2. Reported in: D6.1. Status: completed.

Engage Adapter — Adapter for the Engage collaboration platform. Primary task: T6.2. Reported in: D6.1. Status: completed.

Actyx Integration — Integration of the Actyx middleware into the Babel ecosystem, providing durable and verifiable event broadcast with persistent event logs. Primary task: T6.2. Reported in: D6.2, D6.3 (Section 4.5). Status: completed.

Application-Level Streaming

Decentralised Pub-Sub Streaming — Topic-based publish-subscribe mechanisms for event-driven applications, complementing broadcast-oriented dissemination with application-level filtering and routing. Primary task: T6.3. Reported in: D6.3 (Section 4.6). Status: validated prototype.

Telemetry, Monitoring, and Management

Docker Monitorisation Framework — Centralised telemetry pipeline collecting CPU, memory, network, and storage metrics from nodes and containerised applications, integrated with Prometheus and Grafana. Primary task: T6.3. Reported in: D6.1, D6.2. Status: mature.

Babel-Core Metrics Extension — Protocol-level instrumentation exposing membership, communication, and application-specific metrics in OpenMetrics format from within the Babel runtime. Primary task: T6.3. Reported in: D6.2, D6.3 (Section 4.7). Status: Tested and validated.

Decentralised Telemetry Aggregation — Epidemic and gossip-based aggregation of telemetry data within the swarm, reducing dependence on centralised collection. Primary task: T6.3. Reported in: D6.3 (Section 4.8). Status: working prototype; not yet subjected to rigorous large-scale validation.

ML-Based Self-Management — Emulation environment and initial training pipeline for machine learning models that learn to adjust protocol parameters based on runtime health metrics. Primary task: T6.3. Reported in: D6.3 (Section 4.8). Status: exploratory; closed-loop integration deferred to future work.

Namespace-Based Reconfiguration — Container-based selective reconfiguration of application components using namespace isolation, supporting decentralised and

coordinated updates. Primary task: T6.3. Reported in: D6.2, D6.3 (Section 4.7). Status: validated.

Secure and Privacy-Aware Coordination

Decentralised Energy Market Protocols — Coordination protocols for decentralised resource matching and trade execution, comparing hierarchical and fully peer-to-peer architectures with locality-biased dissemination. Motivated by the EDP renewable energy use case. Primary task: T6.1. Reported in: D6.3 (Section 4.10). Status: exploratory; validated through simulation at scale.

Demonstrators and Integrative Artefacts

TaRDIS Messaging App — Integrative demonstrator exercising the full WP6 protocol stack (membership, gossip broadcast, anti-entropy, storage) across servers, Raspberry Pi, Android phones, and ESP32-based IoT devices, including device-level actuation. Primary task: cross-cutting. Reported in: D6.3 (Section 7.1). Status: operational.

5,000-Node Large-Scale Experiment — Scalability validation of HyParView membership and gossip broadcast at the 5,000-node target, achieving 99.49% average reliability and 1.2s average latency under emulated Internet-scale conditions. Primary task: cross-cutting. Reported in: D6.3 (Section 7.2). Status: completed.

TaRDIS Voting App — Validation demonstrator for Nimbus replicated storage, exercising CRDT-based state management across multiple nodes. Primary task: T6.2. Reported in: D6.2, D6.3 (Section 4.5). Status: completed.

7. Demonstrators and Integrative Artefacts

7.1 TaRDIS Messaging App Demonstrator

The TaRDIS Messaging App is an integrative demonstrator developed within WP6 to provide a concrete executable example of a decentralised, swarm-based application built with the TaRDIS toolbox. Its primary goal is not to deliver a feature-complete messaging product, but rather to exercise and validate the composition of core WP6 mechanisms—namely decentralised membership management, gossip-based dissemination, recovery mechanisms, and heterogeneous device integration—within a single, coherent system.

The application implements a broadcast-oriented communication model in which participants disseminate messages, potentially including attached content, to all nodes in a swarm. In addition to message dissemination, selected nodes are connected to simple IoT devices, allowing the demonstrator to showcase the interaction between decentralised coordination mechanisms and device-level actuation.

7.1.1 Architecture and Deployment Model

The demonstrator is deployed as a decentralised swarm composed of heterogeneous nodes, including Raspberry Pi devices, laptops, mobile devices, and constrained IoT-class hardware. Larger devices execute a Java-based implementation of the application logic built on top of the Babel-Swarm and Babel-Android frameworks, while constrained devices (e.g., ESP32-based M5Stack Core devices) run Micro-Babel and interact with the swarm through lightweight protocols. All nodes operate without central coordination, relying instead on self-organising protocols for discovery and communication

In the demonstrated setup, nodes are connected through a local wireless network for convenience, but the architecture does not fundamentally rely on local connectivity. The swarm model assumes that nodes may join or leave dynamically and that temporary communication failures may occur, reflecting realistic operating conditions for decentralised systems.

7.1.2 Core Protocols and Runtime Components

At the core of the Messaging App lies a decentralised membership service based on a modified version of the HyParView protocol, adapted to leverage the self-configuration capabilities of Babel-Swarm. When a node becomes active, it uses lightweight discovery mechanisms to locate an existing swarm member, which then introduces it into the overlay. The resulting overlay is randomised and resilient, providing the structural foundation for higher-level communication protocols

Message dissemination is implemented using an eager push gossip broadcast protocol that exploits local neighbourhood information supplied by the membership layer. Messages generated by any swarm node are propagated through the overlay, with duplicate reception being expected and explicitly tolerated as part of the robustness–overhead trade-off inherent to gossip-based dissemination. Each node

locally detects and discards duplicates, ensuring that messages are delivered at most once to the application layer

To address transient failures and temporary network partitions, the system incorporates a periodic anti-entropy protocol. This mechanism leverages both neighbourhood information from HyParView and metadata maintained by the broadcast protocol to identify and recover missing messages once connectivity is restored. As a result, the system is able to converge to a consistent dissemination state even in the presence of churn and intermittent disconnections

An additional Random Tour protocol is integrated to provide a decentralised estimation of swarm size. This information is consumed by higher-level application logic and illustrates how monitoring and coordination-related services can be layered on top of the same runtime substrate.

7.1.3 Application Logic and Interaction

Two variants of application logic are demonstrated. In the automated variant, deployed as a Linux service on Raspberry Pi nodes, the application periodically generates messages according to configurable parameters (e.g., probabilistic message generation at fixed intervals). This variant consumes information produced by other protocols, such as swarm size estimates, to illustrate cross-protocol interaction within the runtime.

In the interactive variant, a graphical user interface allows users to send messages and observe swarm activity. The GUI interacts with Babel protocols through a thin wrapper layer that translates user actions into protocol requests and asynchronously receives events related to membership changes and message delivery. This separation highlights the decoupling between application logic and underlying coordination mechanisms promoted by the WP6 architecture

7.1.4 IoT Integration and Heterogeneous Devices

Some swarm nodes are connected to simple IoT devices and managed through a dedicated IoT control component implemented using a set of Babel protocols. This component exposes events and operations for manipulating attached devices and can react to events generated by other protocols, such as the receipt of specific message content. In the demonstrator, this capability is used to trigger visible effects on IoT devices based on keywords embedded in disseminated messages, illustrating end-to-end integration from decentralised communication to physical actuation

Constrained devices running Micro-Babel participate in the swarm by relying on its core networking, discovery, and membership functionalities. Through lightweight connectors, these devices can both receive broadcast messages and issue control commands, demonstrating the feasibility of extending the same coordination abstractions across a wide spectrum of hardware capabilities.

7.1.5. Role within WP6

The TaRDIS Messaging App serves as a reference integration artefact for WP6, validating that the individual mechanisms developed throughout the work package can be composed into a functioning decentralised application. Rather than optimising for performance or usability, the demonstrator emphasises robustness, composability, and

architectural clarity. It provides concrete evidence of how decentralised membership, gossip-based dissemination, recovery mechanisms, monitoring services, and heterogeneous device support can coexist within a unified runtime framework, thereby supporting the broader objectives of WP6 and informing subsequent validation activities in WP7

7.2 Mini Demonstrator for Large-Scale Testing

We built a smaller version of the Demonstrator discussed above to validate its operation in an emulated environment (simulating the Internet) on our computational cluster (NOVA Cluster).

This demonstrator was modified to deal with the limits of RAM in our cluster machines, in practice we used variants of the HyParView and Flood Broadcast protocols that share a single channel, to minimize the number of TCP connections and hence drop CPU and RAM overhead, and we also removed the anti-entropy protocol (that recovers occasional message misses due to instability or readjustments to the overlay topology) as this protocol was the one that consumed more memory.

Our experiments clearly demonstrated that Babel-Swarm (and the developed protocols) could easily scale to 5000 processes in a swarm and achieve adequate performance in message delivery and latency.

7.2.1 Experimental Setup

The experiment was conducted on the NOVA/DI research cluster, a shared computational infrastructure comprising more than 50 heterogeneous physical machines interconnected by high-speed networking. To emulate a large-scale swarm operating over a wide-area network, we deployed 5,000 independent Babel-Swarm processes across 36 of the most powerful machines in the cluster, using the oar-p2p resource reservation system (which was developed in-house) to orchestrate the deployment and a network emulation layer to introduce realistic inter-node latencies and topology constraints representative of Internet-scale communication.

Each process executed the lightweight version of the messaging application described above, thereby composing an instance of the HyParView membership protocol with a flood-broadcast protocol. Both protocols were configured to share a single communication channel, thereby reducing the number of open TCP connections per process and keeping per-node memory consumption within the limits of the cluster hardware. The anti-entropy protocol was deliberately excluded from this configuration, as its periodic state reconciliation-imposed memory requirements were incompatible with the density of processes running on each physical machine. It should be noted that the absence of anti-entropy is expected to negatively affect reliability, since messages missed due to transient overlay readjustments cannot be recovered after the fact.

Each node in the experiment periodically generated broadcast messages, and the system was allowed to run for a sustained 2h of node transmissions, after a sufficient period for all nodes to stabilise their membership views and for a representative volume of messages to be exchanged. Log files were collected from all 5,000 processes and analysed offline to compute per-node reliability (which represents the fraction of processes that deliver a message) and per-message latency (defined as the time

elapsed between the message being broadcast by a process and delivery by the last process that delivered it).

7.2.2 Results and Discussion

The analysis of the collected log files yielded the following aggregate results across all 5,000 processes. A total of 118,175 broadcast messages were generated and disseminated during the experiment. The average reliability across all nodes was 99.49%, with the large majority of messages being delivered by 100% of the processes. The average message delivery latency was 1,232.8 ms, with individual message latencies ranging from 2.0 ms (this was for messages that got only to a small fraction of processes due to readjustments of HyParView overlay topology and the lack of the anti-entropy protocol) to 44,223 ms (which represents the worst case of dissemination we observed in this experiment).

The minimum per-node reliability observed was 16.6%, which is attributable to nodes that experienced transient disconnection during part of the experiment while HyParView was readjusting the topology. This is consistent with the expected behaviour of a system operating without anti-entropy recovery: while the gossip-based broadcast protocol achieves high reliability under stable conditions, nodes experiencing transient disconnections cannot retroactively recover messages disseminated during that period. In a complete deployment, the anti-entropy protocol would address precisely this limitation.

The maximum latency values, while significant in absolute terms, are attributable to the emulated wide-area network conditions and to the multi-hop nature of gossip propagation in large overlays. In a 5,000-node HyParView overlay, messages must traverse several hops to reach all participants, and each hop incurs the emulated latency and processing delays introduced by the cluster environment. The average latency of approximately 1.2 seconds is consistent with the expected behaviour of eager push gossip dissemination at this scale and under these conditions.

Overall, these results provide strong evidence that the Babel-Swarm runtime and the decentralised protocols developed in WP6 can scale to the 5,000-device target established in the TaRDIS project objectives. The combination of high average reliability (above 99%) and bounded latency, even in the absence of recovery mechanisms, validates the effectiveness of the protocol composition and the runtime's robustness under large-scale operation. These findings directly support the assessment of KPIs K-O-4.1 and K-O-4.2 reported in Section 8.

7.3 Role of demonstrators in WP6 validation

Demonstrators played a central role in the validation strategy adopted in WP6, as they provided a practical setting in which core runtime mechanisms were exercised under realistic integration constraints. While a substantial part of WP6 validation was performed through component-level experiments and targeted evaluations reported throughout Section 4, demonstrators complemented this evidence by validating system-level properties that could not be meaningfully assessed in isolation, such as composability

across protocols, end-to-end correctness under dynamic conditions, and operational robustness in heterogeneous deployments.

In particular, demonstrators have served to show that WP6 contributions could be combined into coherent decentralised applications while preserving the intended interaction patterns among membership, dissemination, monitoring, reconfiguration, and data-related components. They also exposed the practical implications of design choices that were difficult to capture with micro-benchmarks, including configuration complexity, dependency management, and the stability of runtime abstractions when integrated with application logic and user-facing interfaces.

The TaRDIS Messaging App demonstrator was representative of this approach, as it exercised decentralised membership management, gossip-based dissemination, duplicate handling, and recovery through periodic anti-entropy, while also validating interoperability across heterogeneous devices and runtime variants (e.g., embedded nodes, edge devices, and mobile platforms). The demonstrator therefore served both as an integration artefact and as a validation instrument, connecting WP6 mechanisms to observable end-to-end behaviour.

Finally, demonstrators also provided an interface between WP6 and WP7 validation activities. Even when a given WP6 component was evaluated independently, demonstrator-driven integration exposed how that component behaved when combined with tools and workflows developed in other work packages. As a result, demonstrators contributed not only validation evidence but also qualitative insights into limitations, integration bottlenecks, and lessons learned, which were reflected in the cross-cutting discussion and the KPI assessment reported in Section 8.

Funded by the European Union

8. KPI Assessment for WP6

8.1 WP6 KPIs

K-O-4.1 — Decentralised membership service

Description: This KPI evaluates the availability of a decentralised membership service capable of operating at a large scale, with deployments of up to 5000 devices. The service must support at least 80% of the devices used by industrial partners and enable dynamic node participation without relying on centralised coordination.

Status: Achieved. WP6 developed multiple variants of HyParView with self-discovery, autonomic management, and security, available across the Babel ecosystem (Babel-Swarm, Babel-Android, and Micro-Babel). The large-scale experiment reported in Section 7.2 demonstrated correct and stable operation with 5,000 concurrent processes, achieving an average reliability of 99.49%. The service operates on servers, Raspberry Pi, Android phones, and ESP32-based devices, covering all device classes used by industrial partners.

K-O-4.2 — Distributed data storage service

Description: This KPI evaluates the availability of a distributed data storage service that supports partial replication and operates at large scale, with deployments of up to 5000 devices. The storage service must support at least 80% of the devices used by industrial partners and enable decentralised data management without central points of failure.

Status: Achieved. WP6 produced three complementary storage solutions: PotionDB for geo-distributed partial replication with materialized views, Arboreal for hierarchical cloud-edge replication with causal+ consistency, and Nimbus as a fully decentralised CRDT-based storage system built on top of Babel-Swarm. Nimbus was validated in the TaRDIS Voting App demonstrator and operates across heterogeneous devices without any dedicated infrastructure. The 5,000-process experiment validates that the underlying communication and membership substrate on which these storage systems operate scales to the required target.

K-O-4.3 — Adapters for external tools and libraries

Description: This KPI evaluates the availability of adapters that enable interoperability between the TaRDIS toolbox and external tools or libraries used by industrial partners. The target is to support at least 50% of the relevant middleware systems, enabling integration with existing infrastructures rather than requiring full replacement.

Status: Achieved. WP6 developed adapters exposing the TaRDIS-defined storage API for Cassandra, IBM Hyperledger Fabric, C3, and Engage, reported initially in D6.1 and maintained through D6.2. The Babel API for Web Services enables integration with applications written in other languages through REST and WebSocket interfaces. The Actyx middleware was also adapted and modularised for integration within the Babel ecosystem.

K-O-5.2 — Programming languages supported by the TaRDIS toolbox

Description: This KPI evaluates the percentage of programming languages used by industrial partners that are supported by the TaRDIS toolbox. The target is to support at least 50% of those languages, ensuring that TaRDIS components can be adopted within heterogeneous development environments.

Status: Achieved. The Babel ecosystem is implemented in Java and natively supports Kotlin through Babel-Android. Micro-Babel extends coverage to C/C++ for constrained embedded platforms. The Babel API for Web Services exposes REST and WebSocket endpoints that allow applications written in any language (including JavaScript, Python, and TypeScript) to interact with the Babel runtime. This covers the programming languages used by industrial partners.

K-O-5.3 — Integration with external middleware and systems

Description: This KPI assesses the degree of integration between the TaRDIS toolbox and external middleware, or systems (such as message brokers or coordination platforms) used by industrial partners. The target is to support integration with at least 50% of such systems, facilitating interoperability and incremental adoption.

Status: Achieved. WP6 integrated the Actyx middleware into the Babel ecosystem, providing reliable, durable event-broadcasting capabilities. Adapters for Cassandra, Hyperledger Fabric, C3, and Engage were developed. The Babel API for Web Services and the namespace-based reconfiguration system (leveraging Docker, Prometheus, and Grafana) provide additional integration points with external middleware and monitoring infrastructure.

K-U-10 — Sub-second latency in live deployments

Description: This KPI evaluates whether TaRDIS solutions can operate live with sub-second latency on deployments of at least twenty nodes. It reflects the runtime performance properties of the decentralised communication and coordination mechanisms provided by WP6.

Status: Achieved. The TaRDIS Messaging App demonstrator operates across heterogeneous devices (Raspberry Pi, laptops, Android phones, ESP32 boards) and delivers messages with latencies well below one second in local network deployments with more than twenty nodes. In the 5,000-node emulated experiment with Internet-scale latencies, the average delivery latency was approximately 1.2 seconds, which is consistent with multi-hop gossip propagation across an emulated wide-area network; in smaller deployments representative of the KPI target, sub-second delivery is consistently achieved.

K-U-11 — Local availability at the device level

Description: This KPI evaluates whether each device in a TaRDIS deployment achieves local availability above 99%. It reflects the resilience and fault-tolerance properties of decentralised services, particularly membership, communication, and storage abstractions provided by WP6.

Status: Achieved. The decentralised design of all WP6 services ensures that each node operates autonomously without depending on central coordination. Membership, communication, and storage abstractions continue to function locally even during network partitions or peer failures. The 5,000-node experiment achieved 99.49% average reliability even without the anti-entropy recovery protocol. In the TaRDIS Messaging App, Android devices operated continuously for at least 48 hours before battery exhaustion, demonstrating sustained local availability.

8.2 WP6 Requirements

All requirements listed below are defined in D2.2 v1.1 – Requirements Analysis and are explicitly assigned to WP6. They specify the functional and non-functional properties expected from the decentralised coordination, communication, storage, telemetry, and configuration mechanisms developed in this work package.

WP6 – General Requirements

RF-WP6-G-01 — Management of cryptographic material

Description: The system must support each participant in managing cryptographic material, enabling the secure operation of decentralised services through the appropriate handling of keys and credentials.

Status: Achieved. Babel-Swarm includes an identity manager that handles self-signed certificates, key generation, digital signatures, and encryption/decryption primitives through Java's Cryptography Architecture and the Bouncy Castle library. The secure variants of HyParView, gossip broadcast, and anti-entropy protocols all leverage this infrastructure.

RF-WP6-G-02 — Support for mobile clients

Description: The system must support mobile clients, including Android devices, enabling them to participate in TaRDIS deployments as full members of the system.

Status: Achieved. Babel-Android provides native support for Android devices, running the Babel runtime as a foreground service to maintain persistent connections and protocol operation. The TaRDIS Messaging App was validated on multiple Android phone and tablet models.

RF-WP6-G-03 — Exactly-once external adapters

Description: External adapters connecting TaRDIS components to third-party systems must guarantee exactly-once semantics, ensuring that interactions with external services do not lead to duplicated or lost operations.

Status: Achieved. The adapters developed for Cassandra, Hyperledger Fabric, C3, and Engage expose a uniform interface that delegates consistency guarantees to the underlying systems. The Babel API for Web Services uses request-reply semantics over REST and WebSockets, preventing duplicated interactions at the application level. Full exactly-once semantics across all failure modes remains dependent on the guarantees of the external system being integrated, but there is no pragmatic solution to circumvent this limitation.

WP6 – Membership Abstractions

RF-WP6-MA-04 — Authenticated decentralised membership abstractions

Description: Membership abstractions must support authentication in a decentralised setting, enabling nodes to verify the identity of participants without relying on a central authority.

Status: Achieved. The secure variant of HyParView uses self-signed certificates for node authentication, with customisable trust policies that are validated during connection establishment via the SecureChannel abstraction in Babel-Swarm.

RF-WP6-MA-05 — Dynamic self-managed overlay networks

Description: The system must support dynamic, self-managed overlay networks that automatically adapt to node joins, departures, and failures.

Status: Achieved. HyParView with self-discovery and autonomic management automatically adapts to node joins, departures, and failures. The protocol dynamically adjusts its active and passive view sizes based on network size estimates provided by the Random Tour protocol.

RF-WP6-MA-06 — Biased dynamic self-managed membership abstractions

Description: Membership abstractions must support biasing mechanisms, allowing preferential selection of peers based on application-specific criteria.

Status: Achieved. X-BOT provides adaptive overlay reshaping based on application-specific criteria such as latency or bandwidth, enabling preferential peer selection while maintaining global connectivity.

RF-WP6-MA-07 — Location-aware dynamic self-managed membership abstractions

Description: Membership abstractions must support location-aware mechanisms, enabling membership decisions that account for geographical or topological locality.

Status: Achieved. WP6 developed location-aware membership abstractions for the EDP use case, departing from the X-BOT biasing technique to promote overlay links between geographically proximate nodes and enable the emergence of location-based communities.

RF-WP6-MA-08 — Isolation between membership abstractions

Description: Different membership abstractions should be isolated from each other, preventing interference between distinct overlays or coordination groups.

Status**:** Achieved. The Babel architecture ensures that each protocol runs in its own execution context with isolated state and dedicated event queues. Multiple membership protocols can coexist within the same runtime without interfering with each other.

RF-WP6-MA-09 — Hierarchical dynamic self-managed membership abstractions

Description: Membership abstractions must support hierarchical structures, enabling multi-level organisation of nodes.

Status: Achieved. WP6 developed membership abstractions supporting hierarchical structures for the EDP use case, enabling multi-level organisation of nodes through both X-BOT-based biasing and Overnesia-derived clique formation with inter-clique hierarchical links.

RF-WP6-MA-10 — Cluster-based dynamic self-managed membership abstractions

Description: Membership abstractions must support cluster-based organisation of nodes to enable scalable and structured grouping.

Status: Achieved. The Overnesia-derived membership protocol organises nodes into cliques that can be mapped to application-level clusters, with inter-clique connectivity ensuring global reachability while preserving cluster-local interaction patterns.

RF-WP6-MA-11 — Administrative domain clusters

Description: Administrative domain clusters must emerge from dynamic self-managed membership abstractions, enabling logical separation of administrative control domains.

Status: Achieved. The hierarchical and location-aware membership abstractions naturally support the emergence of administrative domain clusters, as demonstrated by the community structure developed for the EDP renewable energy market use case.

RF-WP6-MA-12 — Local global information about active elements

Description: Membership abstractions should provide local and global information about active elements in the swarm, enabling nodes to reason about system participation.

Status: Achieved. The Epidemic Global Membership Protocol provides a probabilistic approximation of global membership via epidemic dissemination over partial-view overlays. The Random Tour protocol complements this by providing decentralised estimates of network size.

RF-WP6-MA-13 — Swarm self-configuration

Description: The swarm must be capable of self-configuration, enabling automatic organisation and adaptation without manual intervention.

Status: Achieved. Babel-Swarm supports automatic peer discovery via multicast and broadcast mechanisms, DNS-based parameter retrieval through the @AutoConfigureParameter annotation, and parameter replication from active nodes. Nodes can join a swarm without any preconfigured contact information.

RNF-WP6-MA-14 — Scalable decentralised membership abstractions

Description: Membership abstractions must scale to large numbers of nodes without relying on central coordination mechanisms.

Status: Achieved. The 5,000-process experiment demonstrated that HyParView scales to the required target with 99.49% average reliability and bounded latency, without relying on any central coordination.

RNF-WP6-MA-15 — Always available membership abstractions

Description: Membership abstractions must remain available despite node failures or network partitions.

Status: Achieved. HyParView and its variants operate in a fully decentralised fashion with no single point of failure. The protocol's active/passive view mechanism provides resilience to node failures and ensures that the overlay network remains connected under churn.

WP6 – Communication Abstractions

RF-WP6-CP-16 — Secure point-to-point communication primitives

Description: The system must support secure point-to-point communication primitives for direct interaction between nodes.

Status: Achieved. The SecureChannel abstraction in Babel-Swarm provides encrypted and authenticated point-to-point communication, with certificate exchange and validation during the connection handshake.

RF-WP6-CP-17 — Secure group communication primitives

Description: The system must support secure group communication primitives enabling confidential and authenticated communication within groups.

Status: Achieved. The secure variants of the gossip broadcast and anti-entropy protocols provide encrypted channels, message signing for non-repudiation, and certificate-based authentication for all group communication.

RF-WP6-CP-18 — Dynamic network topologies for communication primitives

Description: Communication primitives must support dynamic network topologies, adapting to changes in connectivity and membership.

Status: Achieved. All communication protocols in the Babel ecosystem operate on top of dynamic overlay networks maintained by membership protocols that continuously adapt to changes in connectivity and membership.

RF-WP6-CP-19 — Isolation between communication abstractions

Description: Different communication abstractions must be isolated from each other to prevent interference between concurrent communication mechanisms.

Status: Achieved. The Babel architecture provides protocol-level isolation through independent execution contexts and event queues, allowing multiple communication abstractions to coexist without interference within the same runtime.

RF-WP6-CP-20 — Reliability of communication primitives

Description: Communication primitives must provide reliable message delivery guarantees.

Status: Achieved. The combination of eager gossip broadcast with anti-entropy recovery provides reliable message delivery. The 5,000-node experiment achieved 99.49% average reliability even without the anti-entropy protocol; with anti-entropy enabled, reliability converges to 100%.

RF-WP6-CP-21 — Low-latency communication mechanisms

Description: Communication mechanisms should support low-latency message delivery.

Status: Achieved. The TaRDIS Messaging App achieves sub-second message delivery in local and small-scale deployments. In the 5,000-node experiment with emulated Internet-scale latencies, the average delivery latency was approximately 1.2 seconds, consistent with multi-hop gossip propagation at that scale. It should be noted that the experiment was executed in an emulated environment where point-to-point links had latency compatible with that observed in the Internet in global-scale peer-to-peer systems (e.g., IPFS).

RF-WP6-CP-22 — Broadcast communication primitives

Description: The system should support broadcast communication primitives for one-to-many dissemination.

Status: Achieved. WP6 provides three broadcast primitives: eager gossip broadcast, flood broadcast, and one-hop broadcast, each with distinct trade-offs between redundancy, reliability, and overhead.

RF-WP6-CP-23 — Epidemic broadcast primitives

Description: The system must support epidemic broadcast primitives enabling scalable dissemination of information.

Status: Achieved. The eager gossip broadcast protocol and its adaptive and secure variants implement epidemic dissemination with configurable fanout, validated at scales up to 5,000 nodes.

RF-WP6-CP-24 — Exactly-once broadcast

Description: Broadcast communication must guarantee exactly-once semantics.

Status: Achieved. All broadcast protocols perform local duplicate detection to ensure at most once delivery to the application layer. Combined with anti-entropy recovery for completeness, the system achieves exactly-once delivery semantics from the application's perspective.

WP6 – Storage Abstractions

RF-WP6-SA-25 — Storage abstractions supporting data integrity and privacy

Description: Storage abstractions should support data integrity and privacy guarantees.

Status: Achieved. Nimbus supports access control policies on keyspaces and collections, allowing creators to restrict read and write access. When combined with Babel-Swarm's secure communication channels, data integrity and privacy are preserved in transit. PotionDB and Arboreal provide CRDT-based conflict resolution, ensuring data integrity under concurrent updates.

RF-WP6-SA-26 — Isolation between storage abstractions

Description: Different storage abstractions must be isolated from each other.
Status: Achieved. Nimbus structures data into keyspaces and collections, each independently configurable. PotionDB uses buckets as the unit of replication. These organisational units provide logical isolation between distinct data sets.

RF-WP6-SA-27 — Dynamic replication

Description: Storage abstractions must support dynamic replication of data.

Status: Achieved. Nimbus supports partial replication at the keyspace and collection levels, allowing nodes to request replication of specific information units at runtime. Arboreal supports dynamic extension of replication towards edge locations. PotionDB allows the system administrator to control where each object is replicated.

RF-WP6-SA-28 — Distributed storage for immutable BLOBs

Description: The system must support distributed storage of immutable binary large objects.

Status: Achieved. The communication and storage infrastructure supports the dissemination and storage of binary content. The TaRDIS Messaging App demonstrated attachment-based file transfer through the broadcast infrastructure, and the Actyx integration provides durable event storage with immutable event logs.

RF-WP6-SA-29 — Distributed storage for mutable key-value pairs

Description: The system should support distributed storage of mutable key-value pairs.

Status: Achieved. Nimbus provides a key-value store interface where values are CRDTs (counters, registers, sets, maps), enabling concurrent mutable updates that converge automatically. PotionDB offers a transactional key-value interface with CRDT-based conflict resolution.

RF-WP6-SA-30 — Low-latency read operations

Description: Storage abstractions must support low-latency read operations.

Status: Achieved. All storage abstractions support local reads on replicated data. Nimbus, PotionDB, and Arboreal serve reads from local replicas without remote coordination, ensuring low-latency access to locally available data.

RF-WP6-SA-31 — High-availability storage abstractions

Description: Storage abstractions should provide high availability.
Status: Achieved. Nimbus operates without any dedicated infrastructure and allows nodes to work offline during network partitions, automatically reconciling state upon reconnection using CRDTs. PotionDB and Arboreal similarly tolerate node failures and partitions, although these last two results have not been employed on any demonstrator or use case implementation as they were not required (they have however, been experimentally validated as reported previously)..

RF-WP6-SA-32 — Durability of storage and logged operations

Description: Storage systems and logged operations must be durable.

Status: Achieved. Nimbus supports data persistence to disk on demand or periodically, enabling recovery from crashes without data loss. The Actyx integration provides durable event storage with persistent event logs.

RNF-WP6-SA-33 — Always available distributed storage abstractions

Description: Distributed storage abstractions must be always available.

Status: Achieved. Nimbus, PotionDB, and Arboreal all operate without central coordination and allow local reads and writes regardless of network conditions. CRDTs ensure eventual convergence without requiring synchronous coordination.

WP6 – Telemetry Acquisition

RF-WP6-TA-34 — Monitoring of memory and communication

Description: The system should monitor memory usage and communication activity of application components.

Status: Achieved. The Docker monitoring and telemetry acquisition framework collects CPU, memory, and network metrics from both nodes and containerised applications. The Babel-core metrics extension enables protocol-level instrumentation of communication activity.

RF-WP6-TA-35 — Durability of communication for auditing

Description: Communication must be durable to support auditing purposes.

Status: Achieved. The telemetry acquisition framework stores collected metrics in a centralised Prometheus time-series database with configurable retention periods. The Actyx integration provides durable and verifiable event logs suitable for auditing.

RF-WP6-TA-36 — Telemetry including membership information

Description: Telemetry acquisition must include information about membership.
Status: Achieved. The Babel-core metrics extension exposes membership-related metrics, including active and passive view sizes and neighbour identities. The Random Tour protocol provides network size estimates that are also collected as telemetry data.

RF-WP6-TA-37 — Telemetry including storage cost

Description: Telemetry acquisition should include information about storage cost for data management.

Status: Achieved. The Docker monitoring framework collects storage-related metrics, including disk usage per node and per container. These metrics are exposed through Prometheus and Grafana dashboards.

RF-WP6-TA-38 — Metrics supporting self-management training

Description: Metrics should be monitored to support the training of self-management mechanisms.

Status: Achieved. The telemetry APIs provide fine-grained stored metrics in OpenMetrics format to machine learning components. An emulated environment was developed to generate training data for ML models that learn to manage protocol parameters based on runtime health metrics such as latency, reliability, and redundant message rate.

RNF-WP6-TA-39 — Scalable telemetry acquisition mechanisms

Description: Telemetry acquisition mechanisms must be scalable.

Status**:** Achieved. WP6 explored epidemic dissemination of telemetry using existing gossip-based communication channels to aggregate metrics within the swarm before transferring them to centralised storage, reducing the volume of data sent to the control plane.

RNF-WP6-TA-40 — Always available telemetry abstraction

Description: Telemetry abstractions must always be available.

Status: Achieved. The Babel-core metrics extension operates locally on each node and does not depend on a centralised collection infrastructure. Metrics are collected and stored locally and can be exported when connectivity is available.

WP6 – Configuration Management

RF-WP6-CM-41 — Dynamic adaptation of memory and communication

Description: The system should dynamically adapt memory consumption and communication patterns of application components.

Status: The autonomic controller in Babel-Swarm dynamically adjusts protocol parameters (e.g., active view sizes, gossip fanout) based on runtime metrics such as network size estimates. The @Adaptive annotation mechanism enables runtime reconfiguration of annotated parameters.

RF-WP6-CM-42 — Replication of essential agents

Description: Essential agents must be replicated to ensure continued operation.

Status: Achieved. Nimbus supports configurable partial replication at the keyspace and collection level, enabling the replication of essential application agents and their state across multiple nodes. The decentralised architecture ensures that no single node is critical for the continued operation of the system.

RF-WP6-CM-43 — Configurable data retention and replication

Description: Data retention and replication policies in distributed data management systems should be configurable.

Status**:** Achieved. Nimbus allows creators of information units to configure and modify replication policies and access control at runtime. The telemetry system supports configurable retention periods for metrics data. PotionDB allows administrators to control where each object and view is replicated.

RNF-WP6-CM-44 — Always available configuration management

Description: Configuration management mechanisms must be always available.

Status: Achieved. The namespace-based reconfiguration system supports decentralised, selective reconfiguration of application components via container-based isolation. The Babel-Swarm autonomic management features operate locally at each node without depending on central coordination.

8.3 Discussion and gaps

The assessment presented in Sections 8.1 and 8.2 shows that all KPIs assigned to WP6 have been achieved and all requirements have been addressed. Nonetheless, a transparent discussion of the nuances, caveats, and remaining open points behind these assessments is warranted.

Regarding scalability, the 5,000-process experiment provides strong evidence that the core membership and communication abstractions developed in WP6 can operate at the target scale defined in the TaRDIS project objectives. However, this experiment was conducted using a streamlined version of the messaging demonstrator (without anti-entropy and with shared communication channels) due to the cluster hardware's physical memory constraints. While this is an engineering limitation of the experimental infrastructure rather than a fundamental limitation of the protocols, a full-stack deployment at this scale — combining membership, gossip broadcast, anti-entropy, storage, and telemetry simultaneously — remains to be demonstrated. The individual components have been validated in isolation and in smaller-scale compositions, and we have no reason to believe that they would not compose correctly at the 5,000-node target, but this has not been explicitly verified end-to-end.

On the topic of latency, the sub-second delivery target (K-U-10) is consistently met in local network deployments with tens of nodes, which matches the KPI definition. At 5,000 nodes under emulated Internet-scale conditions, the average latency was approximately 1.2 seconds, which is a direct consequence of multi-hop gossip propagation over links with realistic wide-area latencies. This is an expected and well-understood behaviour of epidemic dissemination at scale and does not indicate a deficiency in the protocol design. In deployments where nodes are on the same network segment or in the same geographic region, latency remains well below 1 second.

The exactly-once semantics requirement for external adapters (RF-WP6-G-03) is the one item in which achievement is inherently constrained by factors outside WP6's control. The adapters provide clean request-reply interfaces that prevent application-level duplication, but achieving truly end-to-end exactly-once guarantees across arbitrary failures requires cooperation from the external system being integrated. This is a well-known limitation in distributed systems and applies equally to any integration layer; we consider this requirement met to the extent that is pragmatically achievable.

The security mechanisms developed in Babel-Swarm (certificate management, secure channels, message signing) fulfil the relevant requirements but were found during development to expose a more complex API surface than desirable. This observation motivated the ongoing design of Babel 2, which aims to simplify these abstractions and make secure-by-default communication the norm rather than an opt-in feature. The current implementations are functional and validated, but their usability for external developers would benefit from the simplification that Babel 2 is expected to bring.

In the area of storage, Nimbus and the CRDT library have been validated through the TaRDIS Voting App and targeted experiments. PotionDB and Arboreal were validated through their own experimental evaluations reported in Sections 4.5 and 4.4, respectively. However, none of these storage systems have been validated at the 5,000-node scale in isolation; their scalability argument relies on the validated scalability

of the underlying membership and communication substrate. For Nimbus specifically, the partial replication and access control mechanisms are functional but were still undergoing refinement at the time of writing, and a comprehensive performance evaluation at scale is deferred to future work beyond the project.

The telemetry and monitoring contributions satisfy all assigned requirements, but the degree of maturity varies across the different mechanisms. The centralised telemetry pipeline (Docker monitoring, Prometheus, Grafana) is mature and has been used extensively. The epidemic dissemination of telemetry within the swarm has been explored, and a working prototype exists, but it has not been subjected to the same level of rigorous experimental validation as the core communication and membership protocols. Similarly, the machine learning-based self-management work produced a functional emulation environment and initial training results, but the closed-loop integration — where a trained model autonomously manages a live swarm — remains an area for future investigation.

Finally, the ongoing consolidation of the Babel ecosystem into Babel 2 is a recognised gap as of this writing. The existence of multiple partially divergent variants (Babel-Core, Babel-Swarm, Babel-Android) introduced maintenance overhead and occasional compatibility issues, particularly around supported Java versions. Babel 2 is designed to resolve this by converging on Java 17 as the base version and providing a unified codebase. While this consolidation was initiated during the project, it was not completed within the reporting period; therefore, some WP6 components exist in variant-specific implementations. This does not affect the fulfilment of KPIs or requirements, but it does represent an engineering debt that will need to be addressed to ensure the long-term sustainability of the toolbox.

In summary, no fundamental gaps were identified that would prevent the WP6 contributions from being used as intended in TaRDIS use cases and beyond. The caveats discussed above are either well-understood consequences of the operational conditions (e.g., latency at scale), limitations inherent to the problem domain (e.g., exactly-once across external systems), or engineering refinements that are being actively addressed through the evolution towards Babel 2.

Funded by the European Union

9. Exploitation, Sustainability, and Booster Programme

Beyond the technical contributions reported in the previous sections, WP6 has devoted significant effort during the final phase of the project to preparing the exploitation and long-term sustainability of its main results. This effort has been structured around two complementary axes: (i) participation in the Horizon Results Booster (HRB) programme, an initiative of the European Commission operated by META Group and partners, which provides structured go-to-market support to Horizon-funded projects; and (ii) organic adoption of Babel-based technology by external actors in the Portuguese innovation ecosystem. This section reports on both.

The HRB programme was engaged specifically for KER 2 — the Babel Framework and Ecosystem — which was identified as the WP6 result with the highest exploitation potential due to its cross-cutting nature, modular architecture, and applicability to multiple market segments. The programme involved a series of structured workshops and expert-guided exercises, including characterisation of the key exploitable result, identification of a unique value proposition through a Value Proposition Canvas, elaboration of an exploitation roadmap with milestones and cost projections, and a risk assessment and priority mapping exercise. The detailed outputs of these activities are provided in Annex to this deliverable.

9.1 WP6 assets with exploitation potential

The primary exploitable asset in WP6 is the Babel Ecosystem, which encompasses the core Babel runtime, Babel-Swarm (with autonomic, secure swarm execution), Babel-Android (lifecycle-aware mobile execution), and Micro-Babel (embedded and constrained-device support). Together, these components form a unified programming environment that enables developers to create scalable, fault-tolerant, and privacy-preserving IoT and domotics systems capable of operating autonomously, without dependence on centralised cloud infrastructures.

Through the Booster characterisation exercise, the target market for the Babel Ecosystem was refined to focus primarily on European SMEs and startups developing IoT and domotics solutions that require reliability, privacy, and offline functionality. Additional segments include embedded system manufacturers, critical infrastructure operators in energy, transport, and civil protection, and telecom and technology vendors interested in interoperable decentralised platforms. The characterisation process highlighted that end-user pain points — cloud dependence, increasing latency and costs, fragmented development tooling across device classes, difficulty ensuring fault tolerance in resource-constrained environments, and high integration costs that slow innovation cycles — align well with the capabilities offered by the Babel Ecosystem.

The competitive landscape was analysed and found to comprise two main categories of alternatives: cloud operators providing IoT and domotics development abstractions (e.g., AWS IoT, Google Cloud IoT), whose weakness lies in continuous operational costs and mandatory Internet connectivity; and platform-level tool providers (e.g., Home Assistant, Eclipse Kura, AWS IoT Greengrass) whose fragmentation across vendor ecosystems

Funded by the European Union

creates interoperability barriers. The Babel Ecosystem differentiates itself through full decentralisation, cloud independence, unified APIs across heterogeneous device classes, and open-source extensibility. The current Technology Readiness Level of the Babel Ecosystem core components is assessed at TRL 6, with validated prototypes demonstrated in relevant IoT and domotics environments during the TaRDIS project.

The Value Proposition Canvas exercise, conducted during the Booster UVP workshop, resulted in the following unique value proposition statement for the Babel Ecosystem:
"Our unified programming ecosystem helps IoT solution providers who want to build innovative, resilient, and interoperable IoT systems and devices, by reducing over-reliance on cloud and development costs, and increasing autonomy and independence — unlike current cloud-based services."

This formulation was derived from a structured mapping of customer jobs (building interoperable IoT systems, constructing distributed platforms for device acquisition and actuation, providing infrastructure control to end clients), customer pains (cloud dependence, high integration costs, difficulty achieving resilience, tool fragmentation, high innovation costs), and gain creators (rapid deployment of local decentralised systems, cost savings through edge execution, trust through privacy and compliance, and faster innovation via modular and reusable components).

The IP strategy for the Babel Ecosystem adopts a hybrid model. Core framework components will be released under a permissive open-source license (Apache 2.0) to encourage community adoption and contribution, while advanced modules, integration tools, and enterprise-grade features will be offered under commercial licensing terms. Additionally, the filing of a patent covering specific runtime optimisation mechanisms and the decentralised protocol execution model tailored for IoT and domotics is planned, alongside trademark registration for the Babel Ecosystem brand. It should be noted that an Internet Standards Draft describing the general-purpose framework architecture has already been prepared and submitted, which constrains the patent scope to the specific IoT and domotics features, including self-configuration and specialised distributed protocol support. A dual licensing model is foreseen: free use for academic and research purposes, and commercial licenses for industrial exploitation.

9.2 Babel and Micro-Babel as spin-off foundations

The exploitation roadmap developed through the Booster programme outlines a concrete plan for the first six months following the project's completion, centred on consolidating the technical, legal, and organisational foundations required for sustained exploitation.

The planned actions include: finalisation and unification of the open-source core components (Babel, Micro-Babel, Babel-Android, Babel-Swarm) under a coherent governance model between NOVA FCT and the principal researchers who led the design and development of the framework, potentially leading to the creation of a spin-off company partially owned by NOVA FCT and the research team; development of a commercialisation plan defining the hybrid open-source and licensed business model, pricing structure, and partnership strategy for integration and training services; implementation of the IP protection strategy including patent filing and trademark registration; deployment of pilot demonstrations in IoT and domotics contexts to validate interoperability, resilience, and privacy-preserving offline operation; launch of a developer

and adopter community with public repositories, technical documentation, tutorials, and communication channels; preparation of dissemination and promotional materials targeting IoT developers, system integrators, and the domotics sector; and initiation of funding applications for scaling activities through instruments such as the EIC Transition, national innovation grants, or venture capital engagement leveraging the NOVA institutional network.

The exploitation roadmap defines six milestones within the first six months post-project: open-source release of the integrated ecosystem core (month 3), creation of the spin-off (month 4), IP protection established with patent filed and trademark registered (month 5), pilot deployments defined and specified (month 5), business plan and licensing strategy finalised (month 6), and community infrastructure established (month 6). Cost projections estimate approximately 250,000 EUR for the first year — covering developer effort for consolidation and documentation, IP filing, spin-off creation, and dissemination — and 350,000 EUR for years two and three for advanced module development, pilot validation, community management, and ongoing dissemination. Revenue projections are 40,000 EUR in the first year primarily from consulting and training, growing to 300,000 EUR over three years through licensing, technical support contracts, consulting, and training services. Initial coverage is planned through internal research funding from NOVA FCT, applications to European innovation programmes (e.g., EIC Transition), industrial partnerships for co-development and pilot sites, and venture capital investment once the spin-off is established and initial market validation is achieved.

The risk assessment conducted as part of the Booster programme identified and scored risks across five categories: partnership risks, technological risks, market risks, IPR/legal risks, and financial/management risks. The highest-criticality partnership risk is knowledge transfer and key-person dependency, reflecting the concentration of deep technical knowledge in a small core team — a risk mitigated by assigning backup maintainers and enforcing cross-review practices. Technological risks include security vulnerabilities in decentralised communications and performance degradation on constrained IoT devices, both of which are being addressed through the ongoing Babel 2 consolidation effort. Market risks centre on adoption speed and developer onboarding friction, mitigated through pilot deployments, quick start guides, and community engagement. Financial risks stemming from underestimating IP protection costs and delayed return on investment are mitigated through collaborative patent filing, venture capital funding, and an emphasis on consulting and training as faster revenue channels.

Notably, the exploitation pathway described above is not purely prospective. The Babel framework is already being adopted by small companies in Portugal as part of innovation initiatives. In particular, a company called ParadigmShift is actively building a commercial IoT product on top of the Babel and Micro-Babel frameworks, targeting industrial IoT applications in the natural stone sector. This early commercial uptake provides concrete evidence of the framework's readiness for industrial use and validates the exploitation assumptions underpinning the Booster roadmap. Furthermore, two junior researchers who were partially supported by the TaRDIS project — João Brilha and Diogo Fona — contributed to the development of Babel components during their involvement with the project and have continued to engage with the ecosystem beyond their initial research roles, in the context of the activities of ParadigmShift and companies conducting these innovation actions, further reinforcing the sustainability of the technical knowledge base.

9.3 Lessons from transitioning research artefacts to products

The experience gained through the Booster programme and the early adoption of Babel by external actors have yielded several insights relevant to the transition of research artefacts into exploitable products.

First, the importance of architectural coherence for exploitation cannot be overstated. The existence of multiple partially divergent variants of Babel (Babel-Core, Babel-Swarm, Babel-Android) — which was identified as an engineering debt in Section 8.3 — was also flagged during the Booster characterisation as a potential barrier to adoption. From a developer perspective, the ecosystem must present a unified entry point with consistent APIs and documentation. The planned consolidation into Babel 2, converging on Java 17 as the base version, is therefore not only a technical improvement but a prerequisite for viable commercial exploitation.

Second, the Booster exercises made explicit the tension between the breadth of the Babel Ecosystem's potential applications and the need to focus on a specific beachhead market for initial go-to-market efforts. While the framework is technically applicable to a wide range of decentralised and IoT scenarios, the exploitation strategy deliberately narrows the initial focus to IoT and domotics — sectors where the pain points of cloud dependence, fragmentation, and privacy are most acute and where the Babel Ecosystem's differentiators are most immediately valuable. This focus does not preclude expansion into other verticals (e.g., critical infrastructure, telecom) but provides a realistic scope for the first commercial phase.

Third, the dual nature of the exploitation model — combining open-source community building with commercial licensing of advanced modules — requires careful management of the boundary between the two. The Booster IP strategy work highlighted the need to clearly define which components fall under the open-source core and which are reserved for commercial licensing, to avoid ambiguity that could undermine either community trust or commercial viability. The planned use of a permissive open-source license (Apache 2.0) for the core, combined with trademark protection and commercial terms for advanced features, represents a well-established pattern in the industry (cf. approaches used by companies such as Confluent, Elastic, or HashiCorp in their earlier open-source phases) and is expected to provide a workable balance.

Fourth, the early adoption by ParadigmShift demonstrated that the transition from research prototype to commercial product requires not only software maturity but also supporting artefacts such as integration guides, deployment documentation, and reference architectures for specific hardware targets. These supporting materials were not originally within the scope of WP6 but were identified during the exploitation planning as essential enablers for adoption.

Finally, the Booster programme itself proved to be a valuable structured methodology for researchers who are technically proficient but may lack experience in go-to-market planning, business model definition, and IP strategy. The structured exercises — particularly the Value Proposition Canvas and the risk assessment — forced explicit articulation of assumptions that are often left implicit in academic work, such as who the actual customer is, what alternatives they currently use, and why they would switch. This disciplined framing has materially improved the quality and credibility of the exploitation

plan and is a practice that the team intends to continue applying in future research-to-market transitions.

The key documents from the Booster programme are provided as annexes to this deliverable for interested readers (but reading them is not required to understand the work conducted in WP6 in this domain).

Funded by the European Union

10. Lessons Learned and Open Challenges

This section reflects on the technical, integration, and operational lessons accumulated throughout the lifetime of WP6, drawing on the experience of developing, validating, and beginning to transition the toolbox components reported in the previous sections. Rather than a summary of achievements, this section focuses on what was learned from difficulties encountered, trade-offs that proved harder than anticipated, and open problems that remain relevant for future work.

10.1 Technical lessons

The most significant technical lesson from WP6 concerns the tension between the generality of the Babel programming model and the engineering effort required to maintain multiple runtime variants targeting different execution environments. The decision to develop Babel-Swarm, Babel-Android, and Micro-Babel as distinct implementations — each tailored to the constraints of its target platform — was architecturally sound and essential to cover the device heterogeneity envisioned by TaRDIS. However, the practical consequence was the emergence of three partially divergent codebases that shared a common conceptual model but differed in implementation language (Java for the JVM variants, C for Micro-Babel), supported Java versions (Java 22 for Babel-Swarm on servers, Java 17 for Babel-Android due to Android runtime constraints), and the scope of features available on each platform.

This divergence introduced non-trivial maintenance overhead. Bug fixes and improvements to core abstractions — such as changes to the event dispatching model, channel lifecycle management, or protocol registration logic — had to be replicated manually across variants, with each adaptation requiring consideration of platform-specific constraints. In several instances, features developed for Babel-Swarm (such as the autonomic parameter management or the full SecureChannel abstraction) could not be directly ported to Babel-Android or Micro-Babel without substantial redesign. The planned consolidation into Babel 2, converging on Java 17 as the unified base version for the JVM variants, is a direct response to this experience. The lesson is that a multi-platform protocol ecosystem requires not only a shared conceptual model but also a deliberate investment in shared code infrastructure — shared libraries, cross-compilation targets, or code generation — that was not fully in place during the TaRDIS project period.

A second technical lesson relates to the security abstractions in Babel-Swarm. The certificate management, secure channel, and message signing facilities fulfil their functional requirements and were successfully validated in multi-node deployments. However, the API surface exposed to protocol developers proved more complex than desirable. Configuring cryptographic material, establishing trust policies, and enabling secure channels required explicit interaction with multiple subsystems that, in retrospect, could have been unified behind a simpler secure-by-default configuration. This observation was one of the primary motivations for the design of Babel 2, where the goal

is to make encrypted and authenticated communication the default behaviour rather than an opt-in feature that protocol developers must explicitly request and configure.

A third lesson concerns the development of Micro-Babel for constrained devices. The decision to implement Micro-Babel in C on top of FreeRTOS, rather than attempting to run a JVM on microcontroller hardware, was clearly the right choice given the memory and computational constraints of devices such as the ESP32 and RP2040. However, the conceptual gap between the Java-based protocol development workflow used for Babel-Swarm and the C-based workflow for Micro-Babel was larger than initially anticipated. Protocols that had been designed and debugged on the JVM had to be substantially rewritten for Micro-Babel, adapting not only the language but also the memory management strategy (static allocation at build time rather than dynamic allocation), the concurrency model (FreeRTOS tasks with blocking queues rather than cooperative event-driven scheduling), and the communication abstractions (raw radio interfaces for LoRa, BLE, and Wi-Fi rather than TCP/TLS channels). While the event-driven programming model was preserved at the conceptual level — protocols still interact exclusively through events and notifications — the practical effort of porting a protocol from Babel-Swarm to Micro-Babel was closer to a reimplementation than a port. Future work on tooling that could generate Micro-Babel protocol stubs from Babel-Swarm protocol definitions, or a domain-specific language that abstracts over both targets, would substantially reduce this friction.

A fourth lesson emerged from the large-scale experiments conducted on the NOVA/DI research cluster. Running 5,000 concurrent Babel-Swarm processes across 36 physical machines using Docker-based network emulation via oar-p2p revealed several infrastructure-level challenges that are not specific to the protocols themselves but significantly affect experimental methodology. SSH connection rate limiting on the cluster's bastion host required the adoption of SSH multiplexing (ControlMaster) to avoid connection resets when launching processes in parallel. File descriptor exhaustion on cluster nodes, caused by the large number of TCP connections maintained by thousands of concurrent processes, required pre-experiment tuning of kernel limits through modifications to both per-process soft limits and system-wide sysctl parameters. The physical memory constraints of the cluster hardware required the experiment to operate with a streamlined protocol stack — omitting the anti-entropy protocol and sharing communication channels between protocols — to fit 5,000 processes within the available RAM. These are well-understood engineering constraints, but they consumed significant development and debugging time that could have been reduced with better upfront infrastructure characterisation and automated pre-flight checks in the experimental scripts. The scripts developed for these experiments were subsequently adapted for use in the Distributed Algorithms course taught at NOVA, where student groups run smaller-scale experiments on the same cluster infrastructure, providing an additional feedback loop on usability.

10.2. Integration lessons

The integration of WP6 components into the broader TaRDIS system revealed several lessons about the boundaries between composable protocol runtimes and the application-level systems that use them.

The most persistent integration challenge was the interface between the Babel runtime and external systems. The adapters developed for Cassandra, Hyperledger Fabric, C3, and the Actyx event store demonstrated that Babel's event-driven model can be bridged to external APIs through relatively straightforward request-reply wrappers. However, the exactly-once delivery guarantee expected at the adapter interface (RF-WP6-G-03) proved to be inherently limited by the semantics of the external systems being integrated. This is a fundamental property of distributed systems — end-to-end exactly-once semantics require cooperation from all parties — but it surfaced repeatedly during integration work and required careful documentation of what guarantees could and could not be provided. The lesson is that adapter specifications should be explicit about the semantic boundary from the outset, rather than leaving it implicit and discovering the limitation during integration.

The integration between Babel-Swarm and Babel-Android within the same swarm highlighted an unexpected challenge: the two variants, while sharing the same protocol logic, used different Java versions (Java 22 and Java 17 respectively), which introduced subtle incompatibilities in serialisation, cryptographic API behaviour, and network socket handling. These were resolved through careful version-specific branching within the protocol implementations, but the effort involved was disproportionate to the conceptual simplicity of the goal (running the same protocol on a phone and a server). The convergence on Java 17 as the unified base version in Babel 2 is expected to eliminate this class of issues entirely.

The cross-tier integration between Micro-Babel on constrained devices and Babel-Swarm on gateway nodes introduced a different set of challenges. Because Micro-Babel communicates over raw radio interfaces (LoRa, BLE, Wi-Fi) while Babel-Swarm uses TCP/TLS, a bridging layer at the gateway is required to translate between the two transport domains. This bridge must handle not only protocol-level message translation but also connectivity-aware buffering — storing messages locally at the sensor when the gateway is unreachable and forwarding them when connectivity is restored. The Micro-Babel Network Manager provides first-class connectivity awareness for this purpose, generating notification events when link state changes so that protocols can adapt autonomously. While this architecture works well, the design and debugging of the bridging logic was one of the more time-consuming integration tasks in WP6, and the experience suggests that a more structured bridge abstraction within the Babel ecosystem — rather than ad-hoc bridging implemented per deployment — would be a valuable addition for future cross-tier scenarios.

The integration of storage components (Nimbus, PotionDB, Arboreal) with the communication and membership substrate was conceptually clean — all storage systems rely on the same underlying membership and broadcast infrastructure — but

their maturity levels varied significantly at the time of integration. Nimbus was the most tightly integrated with the Babel runtime and was validated through the TaRDIS Voting App, but its partial replication and access control mechanisms were still undergoing refinement. PotionDB and Arboreal were validated primarily through their own standalone evaluations, and their integration into the broader demonstrators was less exercised. The lesson here is that for a toolbox to present a coherent integration story, the maturity of individual components must be managed as a cross-cutting concern, not just a per-component metric.

The Web Services API developed for Babel-Swarm (REST + WebSocket) proved to be an important enabler for integration with non-Java components and external applications. Several use cases that emerged during the project — including the TaRDIS Messaging App's web interface and the integration with monitoring dashboards — relied on this API as the primary interface to the Babel runtime. This suggests that the provision of a well-documented, language-agnostic external API should be treated as a first-class concern in the design of any protocol runtime that aspires to be used in real-world systems, rather than an afterthought added for convenience.

10.3 Scalability and usability trade-offs

The WP6 development experience exposed several trade-offs between scalability, usability, and system complexity that are relevant to the broader design of decentralised systems.

At the scalability end, the 5,000-node experiment demonstrated that the core membership and gossip broadcast protocols scale well to the target specified in the TaRDIS project objectives. The average reliability of 99.49% without anti-entropy, and the average delivery latency of approximately 1.2 seconds under emulated Internet-scale conditions, are consistent with the theoretical properties of epidemic dissemination and provide confidence in the protocol design. However, achieving this scale in practice required careful resource management — bounding per-process memory to 800 MB, limiting heap allocation with explicit JVM flags, and sharing communication channels between co-located protocols — that imposed constraints on the protocol stack that could be deployed. A full-stack deployment at 5,000 nodes — combining membership, gossip broadcast, anti-entropy, storage, and telemetry simultaneously — was not demonstrated within the project, and doing so would require either significantly more physical resources or more aggressive optimisation of per-process memory footprints. This represents a gap between the demonstrated scalability of individual protocol compositions and the envisioned scalability of the complete TaRDIS toolbox operating as an integrated system. This in practice is not a relevant aspect, as we expect that core-nodes in the swarm will have more memory available, even if they are small Raspberry pi devices.

On the usability side, the protocol-centric programming model of Babel has consistently received positive feedback from both researchers within the project and external users,

including students in the NOVA Distributed Algorithms course who use Babel to implement and experimentally compare gossip and membership protocols. The clear separation between protocol logic and system concerns, the explicit event-based interaction model, and the ability to compose and replace protocols without modifying application code are seen as significant advantages over ad-hoc distributed systems development. However, the barrier to entry for new developers remains non-trivial. The Babel API, while conceptually clean, requires familiarity with the event-driven paradigm and with the specific conventions of the framework (protocol registration, handler binding, channel creation) that are not self-evident from the API alone. The documentation and tutorial material developed during TaRDIS and refined through the Distributed Algorithms course have helped to lower this barrier, but more work is needed in the areas of interactive debugging tools, protocol visualisation, and IDE integration to make the framework accessible to developers who are not already familiar with decentralised protocol design.

The experience with Micro-Babel on constrained devices exposed a different facet of the usability trade-off. While the FreeRTOS-based implementation is efficient and well-suited to the target hardware, the development and debugging workflow for embedded C code is substantially more cumbersome than the JVM-based workflow used for Babel-Swarm. The absence of a managed runtime means that common errors such as buffer overflows, use-after-free, and stack overflows can manifest as silent data corruption or hard faults rather than exceptions, making them significantly harder to diagnose. The static memory allocation model, while essential for bounded resource usage, requires developers to size all data structures at compile time, which is less forgiving than the dynamic allocation available on the JVM. Investing in better tooling — including simulation environments where Micro-Babel protocols can be tested on a host machine before being deployed to actual hardware — would substantially improve the development experience for constrained devices.

Finally, the telemetry and monitoring contributions in WP6 illustrate a trade-off between the maturity of centralised and decentralised approaches. The centralised telemetry pipeline (Docker monitoring, Prometheus, Grafana) is mature, well-documented, and was used extensively throughout the project for both development and demonstration purposes. The decentralised alternative — epidemic dissemination of telemetry within the swarm — is conceptually more aligned with the TaRDIS vision of cloud-independent operation but has not been subjected to the same level of rigorous validation. Similarly, the machine learning-based self-management work produced a functional emulation environment and initial training results, but the closed-loop integration where a trained model autonomously manages a live swarm remains an area for future investigation. This reflects a broader pattern observed in WP6: decentralised alternatives to traditionally centralised functions (monitoring, management, storage) are technically feasible and have been prototyped, but achieving the same level of operational maturity requires sustained engineering investment that extends beyond what was possible within the project's timeline.

11. Conclusions and Outlook

This deliverable has reported on the final iteration of the WP6 toolbox components developed within the TaRDIS project. This concluding section summarises the main contributions, positions WP6 within the broader TaRDIS vision, and outlines the directions that will guide future development.

Final WP6 contribution summary

WP6 has delivered a coherent and modular substrate for the development, composition, and execution of decentralised protocols across heterogeneous environments. The centrepiece of this substrate is the Babel ecosystem, comprising Babel-Swarm for general-purpose and swarm-scale deployments, Babel-Android for mobile devices, and Micro-Babel for constrained embedded hardware. Together, these runtimes provide a unified protocol-centric programming model — based on event-driven interaction, explicit protocol composition, and abstract communication channels — that spans the full device spectrum from ESP32 microcontrollers to server-class machines.

On top of this foundational layer, WP6 produced a set of interoperable building blocks addressing the core concerns of decentralised systems: membership management through HyParView and its secure and autonomic variants; reliable message dissemination through gossip broadcast with configurable redundancy and anti-entropy recovery; replicated data storage through Nimbus, PotionDB, and Arboreal, each targeting distinct consistency and replication trade-offs; application-level streaming through decentralised pub-sub mechanisms; and system observability through both centralised and epidemic telemetry approaches. These components were validated individually and in composition, culminating in the 5,000-node large-scale experiment that demonstrated the scalability of the core membership and communication protocols, and in the TaRDIS Messaging App demonstrator that exercised the full protocol stack across servers, Raspberry Pi nodes, and Android devices.

All KPIs that were within the scope of operation of WP6 have been achieved and all requirements have been addressed, as documented in Section 8. The caveats and nuances discussed in Sections 8.3 and 10 are transparently acknowledged but do not compromise the fulfilment of the project objectives.

Position of WP6 within TaRDIS

WP6 occupies a foundational role within TaRDIS. The runtime and protocol abstractions developed in this work package provide the execution substrate upon which the contributions of other work packages depend for communication, coordination, and data management in decentralised settings. The explicit separation between mechanisms and policies — a deliberate architectural choice in the Babel ecosystem — has enabled other TaRDIS partners to build higher-level services on top of WP6 primitives without being constrained by specific protocol choices or deployment assumptions.

The cross-cutting nature of the Babel ecosystem also positions WP6 as the primary bridge between the different device tiers addressed by TaRDIS. The ability to run conceptually identical protocols on servers, mobile phones, and microcontrollers — with each variant adapted to its platform's constraints while preserving the same interaction model — is a distinctive contribution that goes beyond what is typically achieved in research projects focused on a single execution environment. This multi-tier reach is reflected in the demonstrators, in the experimental validation, and in the early industrial adoption reported in Section 9.

Future research and development directions

The most immediate priority beyond the project is the consolidation of the Babel ecosystem into Babel 2 — a unified codebase converging on Java 17 that eliminates the maintenance overhead and compatibility issues arising from the current multi-variant architecture. This consolidation is not merely an engineering cleanup but a prerequisite for sustainable open-source governance and for credible commercial exploitation, as discussed in the exploitation roadmap of Section 9.

Beyond consolidation, several research directions emerge naturally from the work reported in this deliverable. The closed-loop integration of machine learning-based self-management with the Babel runtime — where trained models autonomously reconfigure protocol compositions in response to observed conditions — remains an open and promising direction that was explored but not fully realised within TaRDIS. The design of structured cross-tier bridging abstractions, reducing the ad-hoc effort currently required to connect Micro-Babel sensor networks to Babel-Swarm gateways, would significantly improve the developer experience for IoT deployments. Improvements to the security model, particularly the simplification of the cryptographic API surface towards secure-by-default communication, are already guiding the Babel 2 design. And the development of better tooling for constrained devices — including host-based simulation of Micro-Babel protocols and automated stub generation from Babel-Swarm protocol definitions — would lower the barrier to entry for embedded deployments.

On the exploitation side, the path forward is anchored in the roadmap developed through the Horizon Results Booster programme: open-source release of the unified core, spin-off creation, IP protection, pilot deployments in IoT and domotics, and community building. The early adoption of Babel by companies in Portugal — including the development of a commercial IoT product by ParadigmShift on top of the Babel and Micro-Babel frameworks — provides tangible evidence that the transition from research artefact to industrial tool is not only planned but already underway. Sustaining this momentum will require continued investment in the supporting artefacts identified in Section 10 — documentation, reference architectures, deployment guides, and developer onboarding materials — alongside the core technical evolution of the platform.

The TaRDIS project set out to provide a comprehensive toolbox for decentralised and swarm systems. WP6 has delivered the foundational layer of that toolbox — a protocol runtime, a set of composable building blocks, and an ecosystem that spans from microcontrollers to mobile devices to servers. The work reported in this deliverable

demonstrates that these contributions are technically sound, experimentally validated, and positioned for sustained impact beyond the project's conclusion.

Annexes: Documents produced for the Booster Programme

Characterisation table

KER name: Babel Ecosystem	
	Input from the Beneficiary
Description	The Babel Ecosystem is a unified programming environment for building smart, privacy-preserving IoT and domotics solutions that configure themselves, can operate without a connection to the Internet, and support interoperability across a wide range of devices. Developers taking advantage of Babel Ecosystem and integrators get faster development at lower costs, while end-users get stronger privacy, and solutions that are reliable even when the internet doesn't. Under the hood, Babel provides a wide range of modular components for building reliable and efficient decentralized systems - such as membership abstractions (that dynamically track devices that are part of the system) and reliable communication mechanisms (send once, everyone gets it) - alongside other decentralized protocols that provide powerful abstractions easily to support diverse applications in the IoT and domotics domains.
Target market/end users	The primary market for the Babel Ecosystem includes European SMEs and startups developing IoT and domotics solutions that require, or can benefit from, devising solutions that provide reliability, privacy, and offline functionality - such as smart home controllers, energy efficiency systems, or building automation platforms. The secondary market consists of device manufacturers and system integrators seeking a lightweight, interoperable software-based solution that allows to have a wide range of heterogeneous devices without relying on centralized cloud services or platforms to ensure inter-operation. The Babel Ecosystem addresses a growing need among technology providers and integrators to deliver interoperable, privacy-preserving, cloud-independent, and resilient solutions that ensure that the control and ownership of the system (as well as its data) remain under its exclusive control. According to statista.com, the IoT market in Europe recorded a revenue of over 225 billion U.S. dollars in 2023 and is expected to increase to 411 billion U.S. dollars by 2028. Companies in almost all industries are adopting and investing in IoT solutions to increase efficiency, productivity, and security across their business activities.
End-users needs / problems	- Dependence on cloud services increases latency, operational costs, and privacy risks, which is a major pain for final users. - Fragmented IoT and domotics development tools hinder integration and interoperability, which limits the ability and effectiveness of integrators and applications/systems developers. - Ensuring resilience and fault-tolerance in heterogeneous, resource-constrained systems is complex, which requires developers and integrators to seek specialized support to handle these challenges, which is expensive, and time consuming. - Innovation cycles are slow due to high integration and prototyping costs, which affects integrators and developers on one hand, and limits the ability of end-users to have access to the innovative solutions that they demand.
Competitive advantages	- Lower infrastructure costs by significantly reducing cloud dependence and avoiding vendor lock-in problem, which is a major advantage for developers and integrators that can avoid costs and dependency on cloud infrastructures, and end-users as they benefit from additional privacy and simplicity. - Privacy and sovereignty preserved by running logic closer to final users and their infrastructures, which is a benefit for end-users that retains the ownership of their deployments as well as all data produced/manipulated by it.

	- Competitive advantage through adoption of decentralized, future-proof architectures, which is a significant benefit for integrators and developers. - Faster deployment of decentralized services with reusable frameworks and self-configuration features, which makes the development and integration cycle faster and more effective. - Confidence in system resilience and fault tolerance, which is a benefit for end-users. - Unified APIs and development abstractions across devices (embedded, domotics, mobile) reduce engineering effort and enable faster innovation, which benefits integrators and developers alike.
Use model	The Babel Ecosystem will be exploited following a hybrid model that combines open-source and community-driven, and commercial components. The core libraries and runtime frameworks will be released under a permissive open-source license (e.g., Apache 2.0) to foster transparency, trust, and generate community adoption, allowing the wide community of experiments and thinkers to contribute with their own tools which can further drive innovation and allow us to identify needs and emerging trends. A spin-off company of NOVA FCT will be established to develop specialized components and services that extend the open-source core. These components will be licensed to developers, integrators, and device manufacturers, supporting commercial exploitation in the IoT and domotics domains. Licensing models can be tailored to specific device types or product lines, which allows to avoid the issuing of individual licenses per device that would be impossible to validate since the solution can operate without Internet connections. This model encourages broad adoption by lowering entry barriers for innovators and SMEs while allowing to explore additional revenue sources from consulting, integration, and training services. Over time, the open-source community will drive growth and experimentation, while licensed components will provide reliability and long-term support for production deployments.
Early Adopters	Early adopters of the Babel Ecosystem are expected to include European SMEs and startups developing IoT and domotics solutions that require privacy, interoperability, and offline functionality. These organizations are typically innovation-driven, open to experimentation, and limited by the high costs and rigidity of cloud-centric platforms. Additional early adopters include system integrators and device manufacturers seeking flexible, lightweight, efficient, and correct software solutions to enhance interoperability and long-term maintainability in their products. The Babel spin-off and the open-source community will serve as the main channels to engage these adopters through early access to beta versions, technical support, and opportunities to co-create new modules and new abstractions to foster innovation and enable the creation of solutions for diverse domains. Such collaborations will help validate the technology in real IoT and domotics scenarios, while establishing reference deployments that can server a key demonstrator of the reliability, simplicity, flexibility, and the additional benefits brought by the Babel ecosystem to the broader market.
Adopters' problems/needs	- Over-reliance on cloud: higher latency, high operational costs, privacy concerns, vendor lock-in on solutions that are developed and deployed on client premises, this is a significant limitation for integrators and developers, but also a significant hurdle for end-users that depend on cloud operators and platforms running there, losing sovereignty and control over their deployments.

	- High integration costs and long development cycles for developers of IoT solutions and the high cost (and time) for innovation and exploring new ideas, which negatively affects developers of solutions. - Difficulty ensuring resilience (i.e., fault-tolerance and robustness) in decentralized or resource-constrained environments composed of different heterogenous devices, which is a significant and costly pain for developers and integrators alike. - Fragmentation of tools across embedded, domotics, and IoT development ecosystems, which limits the capacity of integrators to effectively take advantage of different alternatives in the market when building solutions.
Alternative solution	Current solutions heavy rely on cloud-based services, potentially requiring subscription or registration on different platforms that are used by different manufacturers of IoT devices. This solution creates a full dependence on Internet connectivity, third-party operators, cloud-infrastructures, and creates both difficulties in deployment and resilience but also raises concerns related with the privacy of end-users data and exposition of behaviours. One can summarize the current alternative solutions in the specific groups: Cloud-centric platforms (e.g., AWS IoT, Azure IoT, Google Home, Alexa): offer scalability and convenience but create dependence on third-party infrastructures, increase operational costs, and present significant privacy limitations. These systems are incompatible with offline functionality and become unavailable without an Internet connection. Proprietary edge or gateway frameworks (e.g., Cisco solutions, IKEA home solutions): provide partial decentralization but remain tied to vendor-specific hardware or services, reducing interoperability and long-term flexibility. Open-source or DIY frameworks (e.g., Arduino-based solutions): allow experimentation at low cost but often lack adequate tooling and support to build complex systems, provide reliability, and adequate communication abstractions, requiring significant customization effort, and being limited to the devices that can be supported.
Unique Value Proposition	The Babel Ecosystem empowers developers, integrators, and device manufacturers to build smart, privacy-preserving IoT and domotics systems that operate autonomously, offline, and without dependence on centralized cloud infrastructures. Unlike existing frameworks, Babel offers a self-configuring, modular, and decentralized architecture that supports interoperability across diverse devices and environments. Its reusable protocol abstractions — such as membership management and reliable communication — simplify development while ensuring resilience and long-term maintainability. By combining open-source accessibility with robust, production-grade extensions, Babel provides a scalable and sustainable foundation for creating next-generation IoT and domotics solutions that respect user privacy, reduce costs, and keep full control in the hands of those who deploy and use IoT and domotics solutions.
Competitors	The Babel Ecosystem operates within the space of different alternatives for IoT and domotics platforms, each offering relevant strengths but also significant limitations that the Babel ecosystem can address. While solutions developed taking advantage of the Babel ecosystem, there is also space for solutions that can complement and interoperate with them, extending their reach toward decentralized, privacy-preserving, and offline-capable architectures. a) Cloud-based IoT platforms (e.g., AWS IoT Core, Microsoft Azure IoT, Google Home, Alexa) offer excellent scalability, device management, and integration with cloud analytics, many times by having centralized integration with third-party device developers that end-users have to register and interconnect. However, they require

	Internet connectivity and need the availability of centralized infrastructures, which increases operational costs and raises privacy concerns for end-users. Babel can complement these systems by providing the fundamental mechanisms for local, autonomous operation while still inter-operating with services from the cloud to the edge, however in a way that provides additional control for end-users. b) Proprietary edge or home automation ecosystems (e.g., Cisco Solutions, IKEA Home) provide reliable solutions with user-friendly integration but are bound to specific vendors, hardware, or proprietary protocols. Babel's open and modular design still offers the opportunity of interoperability with such systems while avoiding lock-in and supporting custom extensions for broader device compatibility. c) Open-source and DIY solutions (e.g., Arduino) encourage experimentation and rapid prototyping, while empowering innovation at a low cost, but lack resilience and standardization for dependable operation in large-scale or commercial deployments. Babel's modular runtime and reusable distributed protocols can strengthen these ecosystems by adding reliability and decentralized coordination without sacrificing openness and community-driven innovation. By combining openness with robustness, Babel offers a unique bridge between experimental open frameworks and commercial IoT ecosystems, creating new opportunities for collaboration, integration, and co-development across the IoT and domotics landscape, while at the same time offering an alternative to existing solutions that is capable of providing reliability and privacy to end-users in a fully decentralized way that is able to cope with heterogeneous devices.
Timing	The Babel Ecosystem is currently at TRL 6, with its core components — including the runtime, modular communication abstractions, and prototype integrations in IoT and domotics environments — successfully validated in relevant settings. Within the next 12 to 24 months, the focus will be on industrial-grade consolidation and community expansion, guided by the following milestones: • +3 months: Open-source release of the Babel core components and documentation; establishment of the initial developer community. • +8 months: Pilot deployments with IoT and domotics SMEs validating performance, interoperability, and offline operation in real environments, achieving TRL 7. • +10 months: Creation of the NOVA FCT spin-off, focusing on developing and licensing specialized modules for professional use. • +24 months: Transition to TRL 8, with several validated, reliable deployments and a growing community of contributors and adopters. This phased approach ensures a balanced transition from research to exploitation, combining open-source community engagement with the gradual introduction of commercial services. By the time Babel reaches TRL 8, it will be ready for wider market adoption, supported by strong technical validation, a clear IP strategy, and established pilot references.
IP Strategy	The Babel Ecosystem will adopt a **hybrid IP strategy** balancing community-driven innovation with commercial protection: • **Core framework** released under a permissive open-source license (e.g., Apache 2.0) to encourage adoption and community contribution. • **Advanced modules**, integration tools, and enterprise-grade features may be licensed under commercial agreements. • **Trademarks** (e.g., "Babel-Ecosystem" or another) will be registered to protect branding and ensure quality assurance across ecosystem contributions.

	• **Filing a patent for the solution which will be implemented by the computer.** This may be explored for specific runtime optimizations or decentralized protocol mechanisms unique to the Babel ecosystem. • A possibility of **dual licensing model: i)** for academic purposes used for free and; **ii)** commercial licensing exploitation by paid licenses and/or service or collaborative contracts. Licensing will be achieved by commercial agreement to make specific tools and components available for developers and integrators. Licensing for device manufactures will be based on agreements, per component, for specific device types and models, which allows the licensing to be compatible with the fact that the solution does not require Internet connectivity to operate.

booster@meta-group.com
www.horizonresultbooster.eu

BOOSTER

An initiative of the European Commission

The exploitation roadmap

Exploitation roadmap for KER: Babel Ecosystem	
Actions	Within the first six months after the project's completion, NOVA FCT will focus on **consolidating the technical, legal, and organizational basis for the exploitation** of the Babel Ecosystem. The main objectives are to ensure sustainability, secure IP, and prepare for commercial and community-driven adoption in the **IoT and domotics sectors primarily**. **Planned actions:** 1. **Finalize, unify and stabilize the open-source core components** (Babel, Micro-Babel, Babel-Android, Babel-Swarm) and establish their governance between NOVA FCT and the main researchers that have been driving the design and development of Babel, potentially leading to the creation of a spin-off owned partially between these entities. 2. **Develop a commercialization plan** defining the business model (hybrid open-source and licensed modules as well as consulting and training services), pricing structure, and potential partnerships with third-party entities for integration and potentially also for the training services. 3. **Implement the IP protection strategy**: file a patent for the Babel core mechanisms and decentralized protocol execution model; register the "Babel Ecosystem" trademark (it should be noted that a Internet Standard Draft has already been prepared and delivered with the overall idea of the framework for a more general purpose, hence the patent must be focused on specific components and features tailored for IoT and domotics, including self-configuration and specialized support distributed protocols). 4. **Identify Deploy pilots that can be used for demonstration/validation** in IoT and domotics contexts, focusing on interoperability, resilience, and privacy-preserving operation without cloud reliance. 5. **Launch a developer and adopter community**, including public repositories (GitHub), technical documentation, tutorials, and a communication channel for feedback and support, potentially already under the spin-off created for the exploitation of the Babel Ecosystem. 6. **Prepare dissemination and promotional activities**: product one-pager, website, and presentation materials targeting IoT developers and integrators and domotic domains for presentation in fairs and industry/start-up driven events. 7. **Initiate funding applications** for scaling activities (e.g., EIC Transition, national innovation grants, or collaboration with industrial partners, or venture capitalists – taking advantage of the network and brand NOVA as much as possible).

<table>
<tr><td>Roles</td><td>The exploitation of the Babel Ecosystem will be coordinated and led by NOVA FCT, which retains full ownership of the results and is responsible for ensuring their technical evolution, community governance, and commercial valorization. The roadmap foresees a progressive distribution of roles as the ecosystem evolves.
NOVA FCT (IP Owner and Technical Leader)
Maintains and evolves the open-source core components of Babel, ensuring stability, quality, and alignment with research outcomes.
Coordinates the creation of the spin-off company, providing technical guidance and ensuring strategic continuity.
Provides the IP to the Spin-off.
Babel Spin-off (Commercial and Operational Arm)
Leads community building, training, and outreach actions to expand adoption across industry.
Oversees the protection of intellectual property, manages the Babel trademark, and defines licensing terms for commercial modules.
Manages the commercialization of advanced Babel modules and professional services (consulting, integration, certification, and training).
Builds and maintains industrial partnerships with SMEs and device manufacturers to co-develop or license new modules.
Ensures market visibility and brand recognition through communication campaigns and participation in trade and technology events.
SMEs and Early Adopters (Validation Partners)
Participate in pilot deployments during the validation phase to assess Babel’s performance, interoperability, and usability in real IoT and domotics environments.
Provide feedback and co-create extensions or applications that demonstrate Babel’s practical value and potential in different market segments.
Open-Source Community (Collaborative Development)
Contributes to the continuous improvement of the open-source core through feedback, testing, and new module contributions.
Acts as a driver of innovation, ensuring that Babel remains relevant, interoperable, and adaptable to new technological trends.
This clear distribution of roles ensures that NOVA FCT maintains scientific and strategic leadership, while the spin-off and early adopters expand Babel’s impact across industrial and open-source ecosystems.</td></tr>
<tr><td>Milestones</td><td><table>
<tr><td>Milestone</td><td>Timeline</td><td>KPIs / Indicators</td></tr>
<tr><td>M1 – Open-source release of the integrated Babel ecosystem core components</td><td>3 months</td><td>Public repositories available; documentation and examples published;</td></tr>
<tr><td>M2 – Creation of the Spin-off</td><td>4 months</td><td>Spin-off established with clear ownership and transfer of IP from NOVA</td></tr>
</table></td></tr>
</table>

			FCT to the spin-off.
	M3 – IP protection established	5 months	Patent filed and trademark registered
	M4 – Pilot deployments defined	5 months	2 distinct pilots are defined and specified.
	M5 – Business Plan and Commercial Model	6 months	Licensing and pricing strategy defined
	M6 – Community basis established	6 months	Creating of online presence and tools for community drive contributions. >= 1000 Repository visits per month; >= 25 users active per month.
Costs	1st Year – 250,000€ - For developers (consolidation, documentation, preparing pilots), IP filling, creation of spin-off, business plan, and dissemination activities. 120,000€ - Developers and technical staff 100,000€ - IP filling and creation of spin-off 30,000€ - Dissemination activities 2nd and 3rd Year – 350,000€ - For developers (advanced modules to be licensed, improvements, interactions with community), maintenance, dissemination. 210,000€ - Developers and technical staff €50,000 – Pilot development and validation: equipment, licenses, and integration tools for demonstration sites. 60,000€ - Community Management 30,000€ - Dissemination activities		
Revenues	0-1 year – 40,000€ - Mostly from consulting and training. 40,000€ - Consulting 1-3 year – 300,000€ - Licencing, technical support contracts, consulting, training. 50,000€ - Training Activities 100,000€ - Consulting 150,000€ - Licensing		
Other sources of coverage	**Internal research funding from NOVA FCT** will provide initial support during the first six months after project completion, covering the costs of spin-off preparation and technical consolidation. This support includes granting free ownership of the IP in exchange for NOVA's participation in the spin-off, as well as the allocation of technical staff for initial diligence, setup, and documentation efforts. **Applications to European innovation programs** (e.g., EIC Transition or equivalent Horizon Europe innovation actions) are planned within the first 6 to 9 months post-		

	project, aiming to co-finance the unification and consolidation of the existing Babel components and to support the identification of early adopters for commercial case studies. These applications will align with the roadmap's first validation phase (M9 milestone) to ensure continuity of development and pilot deployment. **Partnerships with industrial stakeholders** in the IoT and domotics sectors will be pursued from the first year onward, both to provide pilot sites and to co-develop specialized modules that demonstrate Babel's practical value. These collaborations will serve as the foundation for early licensing opportunities and as validation cases for future funding proposals and should be established in 12 months. **Venture capital investment** will be sought in the second year, once the spin-off is established and initial market validation has been achieved, to support scaling operations, marketing, and team expansion. Funding timelines may introduce uncertainties, particularly regarding open calls for funding that have variable response times. To mitigate these risks, NOVA FCT will ensure operational continuity through internal support and ongoing collaborations, maintaining essential development and moving forward with community establishment activities while awaiting external funding results. Contingency actions include adjusting pilot deployment schedules and accelerating contacts with industrial partners for co-funded demonstration building activities if external funding is delayed.

booster@meta-group.com
www.horizonresultbooster.eu

BOOSTER

An initiative of the European Commission

Business Plan for Spin-off NOVA Edge

Executive Summary

NOVA Edge is a technology spin-off of NOVA School of Science and Technology (FCT NOVA) dedicated to commercialising the Babel Ecosystem, a programming environment for building autonomous, privacy-preserving, and decentralised IoT and domotics systems. The ecosystem enables devices to operate locally - even in the absence of Internet connectivity - addressing growing demand for resilience, data sovereignty, and cloud-independent operation.

The company will adopt a hybrid open-source and commercial model. The core of Babel will be freely available, fostering a global developer community, while NOVA Edge will generate revenue from licensing advanced modules, integration and consulting services, training programmes, and participation in collaborative research and innovation projects.

The financial plan anticipates approaching the break-even at the end of Year 4, and effectively reaching it in Year 5 through controlled cost growth, diversified revenue streams, and efficient use of external funding combined with internal revenue. The company will operate with lean but scalable teams and will benefit from NOVA FCT's in-kind contributions, including access to laboratories, infrastructure, and innovation support services.

Over five years, NOVA Edge aims to become a leading European supplier of decentralised IoT infrastructure, achieving a turnover above €700,000 in Year 5 and establishing long-term industrial partnerships.

The Organisation

NOVA Edge will be majority-owned by NOVA FCT, ensuring alignment with scientific objectives and continuity in long-term research. The management board will include a representative of NOVA FCT and the CTO, Co-Technical Lead, and Product Manager of NOVA Edge.

Technical leadership is provided by João Leitão (Associate Professor, inventor of widely adopted peer-to-peer protocols) and Nuno Preguiça (Full Professor, globally recognised for CRDTs). Engineering capacity comes from Diogo Jesus (advanced module design), João

Brilha (Micro-Babel lead developer), and Rafael Matos (Babel next-generation architecture lead).

Operational responsibilities will be organised around:

- Babel Core and Integrated Devices
- Protocols and System Integration
- Community and Ecosystem Development
- Business Development and Sales

NOVA FCT's IRIS office will support administrative, legal, and technology-transfer functions. NOVA Edge will adopt a participatory and transparent decision-making model with structured quarterly reviews and long-term governance stability.

Product and Service Description

The Babel Ecosystem provides a modular, decentralised computing environment for IoT and domotics systems. It enables devices to operate autonomously without Internet connectivity, offering resilience, low latency, privacy, and interoperability across heterogeneous devices.

Main components include:

- **Babel Core**: runtime and modular distributed protocols (Java-based)
- **Micro-Babel**: C implementation for embedded systems
- **Reusable protocol modules** for membership, broadcast, coordination, decentralized storage, among others
- **Advanced commercial modules** under development (security, interoperability, fault-tolerance)

Commercial services will include:

- Integration and consulting
- Specialised training and certification
- Long-term support contracts
- Participation in R&D projects

The ecosystem is currently at TRL 6, expected to reach TRL 7 through pilots within three months and TRL 8 within six months of spin-off creation.

Financial Plan

The financial plan presents a detailed projection of NOVA Edge's costs, revenues, liquidity position and investment needs over the first five years of operation. It demonstrates how the company will evolve from an initial investment phase to operational sustainability, reaching approaching break-even in Year 4 and achieving stable profitability in Year 5. All financial projections are aligned with the operational strategy and market assumptions described in previous sections, while incorporating conservative estimates and explicit treatment of in-kind contributions, founders' capital, and the risks associated with project funding delays.

Financial Objectives

Over the first five years, NOVA Edge aims to:

- Reach break-even by the fourth year of operation
- Reach a turnover above €700,000 in Year 5
- Maintain a gross margin above 50% from Year 4 onward
- Achieve an EBITDA margin of around 10% by Year 5
- Maintain liquidity above 10% of yearly expenditure
- Demonstrate long-term sustainability through diversified revenue streams
- Reduce operational risk by relying on a mix of R&D project funding, consulting, training and licensing revenues

Cost Structure Assumptions

All personnel costs follow Portuguese labour law, including **14 months of salary per year** and employer social-security contributions (23.75%). CTO and co-CTO contributions appear both as **costs** and **in-kind institutional contributions** (reflecting NOVA FCT's commitment).
Personnel accounts for roughly 70% of total expenditure, which is aligned with tech startups.

Personal Costa

The following table presents the plans for hiring and cost of employees. Except for the three junior engineers to be hired at the start of year three, year four, and year five, respectively, all personnel will be hired when the company is created.

For the faculty involved in the project (João Leitão and Nuno Preguiça) we assume that they will dedicate 15% of their time to the management of NOVA Edge, that will be contributed by NOVA FCT in-kind. For completeness the cost of their time is also considered here.

Role of Employee	Annual Salary (€)	Employer Additional Taxes (social security)	Employee Annual Total Cost (€)
Chief Technical Officer (João Leitão) – 15%	8302	23.75%	10274
Co-Chief Technical Officer (Nuno Preguiça) – 15%	10752	23.75%	13306
Product Manager (Diogo Jesus)	50000	23.75%	61875
Senior Engineer (João Brilha)	40000	23.75%	49500
Senior Engineer (Rafael Matos)	40000	23.75%	49500
Community Manager (to be hired)	28000	23.75%	34650
Sales Manager (to be hired)	65000	23.75%	80437
Junior Engineer (to be hired at start of year 3)	35000	23.75%	43312
Junior Engineer (to be hired at start of year 4)	35000	23.75%	43312
Junior Engineer (to be hired at start of year 5)	35000	23.75%	43312

Below we provide a summary of the personnel costs per year of operation.

Year	Filled Positions	Total Personnel Costs (€)
Year 1	CTO, Co-CTO, Product Manager, two senior engineers, community manager and sales manager	299542
Year 2	Same as year 1	299542
Year 3	Same as year 2 + 1 Junior Engineer	342854
Year 4	Same as year 3 + 1 Junior Engineer	386166
Year 5	Same as year 4 + 1 Junior Engineer	429478

Operational Costs

Operational costs represent all non-personnel recurring expenses necessary to maintain NOVA Edge's activity. These values remain stable across the first five years, since office space, cloud infrastructure, and support services do not scale significantly with the early team size.

Type of Cost	Annual Cost (€)
Office/lab space	5000
Cloud services, hosting, open-source repositories	4000
Stationary, Printing, Equipment depreciation	3000
Access to testbed equipment	10000
Accounting + Auditing Services	6000
Legal Support + IP Protection	10000
Marketing and Dissemination Activities	30000
Total operational costs per year	68000

Valuation of In-Kind Contribution of NOVA FCT

To ensure the viability of the company, NOVA FCT (major owner of the spin-off) will contribute in-kind. For completeness of this financial report, we quantity the values of those contributions (this is based on estimates considering usual market values for some services). Values reported are per year and we do not expect these values to change significantly during the first five years of operation.

Contribution	Estimated Value (€)
Use of Laboratory and Office Space	5000
Access to testbed equipment	10000
IRIS Administrative and Legal Support	16000
Time of CTO (João Leitão) – 15%	10274
Time of Co-CTO (Nuno Preguiça) – 15%	13306
Total In-Kind Contribution per Year	54580

Founder Capital Contribution

Founder	Initial Capital Contribution (€)	Investment Fraction
CTO (João Leitão	5000	33.33%
Co-CTO (Nuno Preguiça)	5000	33.33%
Product Manager (Diogo Jesus)	2500	16.67%

Senior Engineer (João Brilha)	1250	8.33%
Senior Engineer (Rafael Matos)	1250	8.33%
Total Founder Capital	15000	100%

At incorporation, NOVA Edge will receive €15,000 in initial paid-in capital from the founding team. These contributions demonstrate commitment to the project. Contributions are as follows: €5,000 from the CTO, €5,000 from the Co-CTO, €2,500 from the Product Manager, and €1,250 each from the two Senior Engineers. This initial capital supports early liquidity and complements institutional in-kind contributions from NOVA FCT.

Revenue Model

Below we describe the forecast for the revenues of NOVA Edge for the first 5 years of operation. In doing this prediction we attempted to be conservative regarding the revenue success of the company.

For this analysis we have considered four main revenue streams:

- Licensing of advanced Babel modules
- Consulting and Integration Services
- Training and Certification Activities
- Research and Development Project Participation

Licensing Revenue Forecast

Here we consider the revenue generated by licensing of Babel specialized modules. We divide this source of revenue in two components, licensing for small and medium enterprises, which we consider as valuable partners for the development of our business model, that we consider 5000€ per license. We consider also licensing to larger enterprises, here we consider a cost per year of 20000€.

Year	SME Licenses (€5K)	Enterprise Licensing (€20K)	License Revenue over the year (€)
Year 1	0	0	0
Year 2	1	0	5000
Year 3	2	1	30000
Year 4	4	2	60000
Year 5	6	3	90000

Consulting, Training, and Research Projects Revenue

In the following we make (conservative) projections for the revenue of NOVA Edge in the five first years of operation considering a diversified set of activities. Consulting activities to setup projects for specific clients; training activities to teach how to use the technology for developers; and finally, participation in financed R&D projects. It should be noted that it should be fairly easy to participate in a R&D project in the first year of activity, however, since the calendar for calls (particularly for European projects) might not align with the creation of the startup (and time required to register a PIC) we decided to consider that in the first year of operation NOVA Edge will not be participating in any research and development project.

Year	Consulting (€)	Training (€)	R&D Project Participation (€)	Total Revenue over the year (€)
Year 1	30000	10000	0	40000
Year 2	60000	20000	65000	145000
Year 3	80000	25000	130000	235000
Year 4	120000	35000	215000	370000
Year 5	150000	40000	265000	455000

Summarization of Revenue

Below we provide a summarization of the revenue considering all revenue sources.

Year	Licensing (€)	C/T/R&D (€)	Total Revenue (€)
Year 1	0	40000	40000
Year 2	5000	145000	150000
Year 3	30000	235000	265000
Year 4	60000	370000	430000
Year 5	90000	455000	545000

Projected Income

In the following we merge the several sources of costs and predicted revenue to analyse the evolution of the project income of NOVA Edge. This analysis allows us to conclude that the break-even point will be achieved by year five of operation.

Year	In-Kind contributions (€)	Personal Costs (€)	Operational Costs (€)	Revenue (€)	EBITDA (€)	EBITDA Margin	Net Result (€)

Year 1	54580	299542	68000	40000	-327542	-	-327542
Year 2	54580	299542	68000	150000	-217542	-	-217542
Year 3	54580	342854	68000	265000	-145854	-	-145854
Year 4	54580	386166	68000	430000	-24166	-5.6%	-24166
Year 5	54580	429478	68000	545000	+47522	8.7%	+47522

Cash Flow Analysis

Cash flow analysis is essential for anticipating liquidity needs, mitigating financial risk, and ensuring NOVA Edge maintains operational continuity even under variable revenue conditions. Given the structure of revenues—particularly the dependence on consulting contracts, training cycles, and R&D project reimbursements—cash inflows may be irregular, especially during the early years.

Cash Flow Projections (Years 1–2, Monthly)

The first two years of operation are characterised by negative cash flow, as NOVA Edge invests heavily in core personnel and foundational operational costs before revenues reach a sustainable level. Consulting and training activities start generating income from Year 1, and R&D project participation becomes an important driver of revenue from Year 2 onwards. In parallel, in-kind contributions from NOVA FCT and an initial founders' capital injection of €15,000 help reduce the effective burden on the company, although they do not remove the need for careful liquidity management.

Key characteristics of Years 1–2 cash flow:

- Average monthly outflow in Year 1 of approximately €27,000 (with peaks in July and November due to double salary payments).
- Average monthly inflow in Year 1 of approximately €3,333.
- Average monthly inflow in Year 2 of approximately €12,500, reflecting the growth of consulting, training, and R&D revenues.
- Liquidity coverage supported by €54,580/year of in-kind contributions from NOVA FCT and €15,000 in founders' capital, which together provide an important buffer but do not fully compensate the negative cash flow in the first two years.

- The main financial risk in this period is the absence of R&D income in Year 1 and the dependency on successfully securing and executing projects in Year 2.

Year 1 - Monthly Cash Flow Overview

- Total annual outflow: €367,542
  - Personnel: €299,542
  - Operational: €68,000
- Total annual inflow: €55,000
  - Revenue: €40000
  - Founders' Capital €15000
- Net annual cash variation: –€312,542

Year 1 uses the 14-month salary structure: each employee receives 14 salary instalments per year, with double payments in July and November. This creates two months with significantly higher personnel costs and therefore deeper negative cash flows.

Month	Founders Capital (€)	Revenue (€)	Personal Costs (€)	Operational Cost (€)	Total Outflow (€)	Net Cash Flow (€)
January	15000	3333	21395.86	5666.67	27062.53	-8729.53
February		3333	21395.86	5666.67	27062.53	-23729.20
March		3333	21395.86	5666.67	27062.53	-23729.20
April		3333	21395.86	5666.67	27062.53	-23729.20
May		3333	21395.86	5666.67	27062.53	-23729.20
June		3333	21395.86	5666.67	27062.53	-23729.20
July		3333	42791.71	5666.67	48458.38	-45125.05
August		3333	21395.86	5666.67	27062.53	-23729.20
September		3333	21395.86	5666.67	27062.53	-23729.20
October		3333	21395.86	5666.67	27062.53	-23729.20
November		3333	42791.71	5666.67	48458.38	-45125.05
December		3333	21395.86	5666.67	27062.53	-23729.20

Year 2 - Monthly Cash Flow Overview

Year 2 benefits from the start of R&D project participation and growth in consulting and training revenues. However, cash flow remains negative and requires continued external financing.

- Total outflow: €367,542
- Total inflow: €150,000
- Net annual cash variation: –€217,542

Month	Revenue (€)	Personal Costs (€)	Operational Cost (€)	Total Outflow (€)	Net Cash Flow (€)
January	12500	21395.86	5666.67	27062.53	-14562.53
February	12500	21395.86	5666.67	27062.53	-14562.53
March	12500	21395.86	5666.67	27062.53	-14562.53
April	12500	21395.86	5666.67	27062.53	-14562.53
May	12500	21395.86	5666.67	27062.53	-14562.53
June	12500	21395.86	5666.67	27062.53	-14562.53
July	12500	42791.71	5666.67	48458.38	-35958.38
August	12500	21395.86	5666.67	27062.53	-14562.53
September	12500	21395.86	5666.67	27062.53	-14562.53
October	12500	21395.86	5666.67	27062.53	-14562.53
November	12500	42791.71	5666.67	48458.38	-35958.38
December	12500	21395.86	5666.67	27062.53	-14562.53

Year 3 - Quarterly Cash Flow

- Revenue: €265,000 (€66,250 per quarter)
- Personnel costs: €342,854 (quarterly distribution accounts for July & November double payments)
- Operational costs: €68,000 (€17,000 per quarter)
- Net cash variation (annual): –€145,854

Quarter	Revenue (€)	Personnel Costs (€)	Operational Costs (€)	Total Outflow (€)	Net Cash Flow (€)
Q1	66250	73469	17000	90469	-24219
Q2	66250	73469	17000	90469	-24219
Q3	66250	97958	17000	114958	-48708
Q4	66250	97958	17000	114958	-48708

Year 4 - Quarterly Cash Flow

- Revenue: €430,000 (€107,500 per quarter)
- Personnel costs: €386,166 (quarterly distribution accounts for July & November double payments)
- Operational costs: €68,000 (€17,000 per quarter)
- Net cash variation (annual): –€24,166

Quarter	Revenue (€)	Personnel Costs (€)	Operational Costs (€)	Total Outflow (€)	Net Cash Flow (€)

Q1	107500	82750	17000	99750	+7750
Q2	107500	82750	17000	99750	+7750
Q3	107500	110333	17000	127333	-19833
Q4	107500	110333	17000	127333	-19833

Year 5 - Quarterly Cash Flow

- **Revenue:** €545,000 (€136,250 per quarter)
- **Personnel costs:** €429,478 (quarterly distribution accounts for July & November double payments)
- **Operational costs:** €68,000 (€17,000 per quarter)
- **Net cash variation (annual):** +€47,522

Quarter	Revenue (€)	Personnel Costs (€)	Operational Costs (€)	Total Outflow (€)	Net Cash Flow (€)
Q1	136250	92031	17000	109031	+27219
Q2	136250	92031	17000	109031	+27219
Q3	136250	122708	17000	139708	-3458
Q4	136250	122708	17000	139708	-3458

Break-Even Analysis

The break-even analysis identifies the moment at which NOVA Edge's operating revenues become sufficient to cover all recurring personnel and operational costs, excluding non-cash in-kind contributions. Based on the current financial projections, the company approaches operational break-even during the fourth year of activity. In that year, revenues reach €430,000, while personnel and operational costs total approximately €454,000, resulting in a small operating deficit of €24,166. This value is sufficiently close to equilibrium to indicate that the business becomes structurally sustainable, with all indicators suggesting that even a modest increase in licensing, consulting, or R&D participation would eliminate the remaining gap.

A more granular examination of cash flows reveals that break-even is reached between the first and second quarters of Year 4, when quarterly revenues finally begin to match or exceed quarterly expenditure except in months where double-salary obligations apply. Once Year 4 revenue streams stabilise, the company enters a positive operating cycle, generating the first positive EBITDA during Year 5. Although cumulative liquidity remains negative due to substantial up-front investment and early operating deficits, the business model demonstrates that NOVA Edge becomes operationally viable from Year 4 onward

and capable of generating profit in Year 5. This trajectory is typical of research-based deep-tech ventures, where significant early investment precedes revenue acceleration.

Cash Flow Projections for Years 3–5 (Quarterly)

From Year 3 onward, cash-flow projections are computed on a quarterly basis to provide a clear understanding of seasonal financial dynamics, particularly the impact of Portugal's double-salary months and the progressive growth of revenue streams. The company continues to rely on external funding in Year 3, although this year marks the beginning of a more balanced revenue mix incorporating licensing, consulting, training, and participation in multiple R&D projects.

In Year 3, quarterly cash flow remains negative throughout the year, declining from approximately –€530,000 at the end of Year 2 to around –€650,000 at the end of Year 3.". Despite this, Year 3 represents a turning point in terms of revenue maturity, as consulting and R&D income begin to offset a substantial portion of personnel and operational costs. By Year 4, cash outflows decrease relative to inflows as the company enters its break-even region. Although liquidity remains negative, reflecting accumulated deficits from earlier years, quarterly losses shrink significantly. In the first two quarters of Year 4, the company operates within a narrow margin of equilibrium, while the third and fourth quarters show modest negative cash flow driven primarily by double-salary obligations.

In Year 5, NOVA Edge generates a positive annual EBITDA for the first time and achieves a fully profitable operating cycle. Quarterly results alternate between small surpluses in normal-salary months and modest deficits in double-salary periods, but the annual result remains positive. While liquidity does not return to a positive balance within the five-year window due to earlier accumulated deficits, the company reaches a structurally sustainable financial position. It begins the process of rebuilding liquidity reserves, placing itself in a strong position for future expansion, investment attraction, or entry into new markets.

Sensitivity Analysis

A sensitivity analysis was conducted to evaluate the resilience of NOVA Edge's financial model under different risk scenarios, focusing on key variables that could significantly affect revenue and expenditure. Three major scenarios were examined: (i) the failure to secure an expected R&D project, (ii) a six-month delay in revenue from licensing activities, and (iii) a 10% overrun in personnel costs due to labour market pressure.

The first scenario examines the non-award of an R&D project, which would result in a reduction of approximately €75,000 in yearly revenue. This reduction impacts all years from Year 2 onward and would delay operational break-even beyond Year 4. While this represents the most critical single risk factor, the model remains recoverable through practical mitigation strategies such as delaying a junior engineering hire, increasing the number of SME licences, or securing additional national industry-funded innovation contracts. Any of these adjustments would be sufficient to restore the break-even point to Year 4.

The second scenario considers a delay in licensing revenue, assuming licences are secured only in the second half of each year. This scenario has a moderate impact on financial performance, reducing licensing income by 30–50% in affected years. Nevertheless, the company would still be able to maintain the operational break-even point in Year 4 due to the strength and reliability of consulting and R&D revenue streams. Small adjustments in service pricing or acquisition of additional consulting engagements would further stabilise this trajectory.

The third scenario evaluates a 10% increase in personnel costs, which would raise annual expenditure by €30,000–€40,000. This situation would delay operational break-even by approximately one year. However, the model demonstrates that deferring a single junior hire or increasing the number of R&D projects can neutralise the impact entirely. Given the high likelihood of securing multiple R&D projects under Horizon Europe and national funding schemes, NOVA Edge retains robust capacity to absorb such fluctuations.

Taken together, these scenarios demonstrate that NOVA Edge's financial model is resilient and adaptable. While early years rely heavily on external funding and controlled expenditure, the combination of diversified revenue streams, conservative cost assumptions, and flexibility in recruitment ensures the company's ability to maintain a break-even trajectory and achieve long-term financial sustainability.

Financial Plan for Spin-off NOVA Edge

Executive Summary

NOVA Edge is a technology spin-off of NOVA School of Science and Technology (FCT NOVA) dedicated to commercialising the Babel Ecosystem, a programming environment for building autonomous, privacy-preserving, and decentralised IoT and domotics systems. The ecosystem enables devices to operate locally - even in the absence of Internet connectivity - addressing growing demand for resilience, data sovereignty, and cloud-independent operation.

The company will adopt a hybrid open-source and commercial model. The core of Babel will be freely available, fostering a global developer community, while NOVA Edge will generate revenue from licensing advanced modules, integration and consulting services, training programmes, and participation in collaborative research and innovation projects.

The financial plan anticipates approaching the break-even at the end of Year 4 and effectively reaching it in Year 5 through controlled cost growth, diversified revenue streams, and efficient use of external funding combined with internal revenue. The company will operate with lean but scalable teams and will benefit from NOVA FCT's in-kind contributions, including access to laboratories, infrastructure, and innovation support services.

Over five years, NOVA Edge aims to become a leading European supplier of decentralised IoT infrastructure, achieving a turnover above €700,000 in Year 5 and establishing long-term industrial partnerships.

The Organisation

NOVA Edge will be majority-owned by NOVA FCT, ensuring alignment with scientific objectives and continuity in long-term research. The management board will include a representative of NOVA FCT and the CTO, Co-Technical Lead, and Product Manager of NOVA Edge.

Technical leadership is provided by João Leitão (Associate Professor, inventor of widely adopted peer-to-peer protocols) and Nuno Preguiça (Full Professor, globally recognised for CRDTs). Engineering capacity comes from Diogo Jesus (advanced module design), João Brilha (Micro-Babel lead developer), and Rafael Matos (Babel next-generation architecture lead).

Operational responsibilities will be organised around:

- Babel Core and Integrated Devices
- Protocols and System Integration
- Community and Ecosystem Development
- Business Development and Sales

NOVA FCT's IRIS office will support administrative, legal, and technology-transfer functions. NOVA Edge will adopt a participatory and transparent decision-making model with structured quarterly reviews and long-term governance stability.

Product and Service Description

The Babel Ecosystem provides a modular, decentralised computing environment for IoT and domotics systems. It enables devices to operate autonomously without Internet connectivity, offering resilience, low latency, privacy, and interoperability across heterogeneous devices.

Main components include:

- **Babel Core**: runtime and modular distributed protocols (Java-based)
- **Micro-Babel**: C implementation for embedded systems
- **Reusable protocol modules** for membership, broadcast, coordination, decentralized storage, among others
- **Advanced commercial modules** under development (security, interoperability, fault-tolerance)

Commercial services will include:

- Integration and consulting
- Specialised training and certification
- Long-term support contracts
- Participation in R&D projects

The ecosystem is currently at TRL 6, expected to reach TRL 7 through pilots within three months and TRL 8 within six months of spin-off creation.

Financial Plan

The financial plan presents a detailed projection of NOVA Edge's costs, revenues, liquidity position and investment needs over the first five years of operation. It demonstrates how the company will evolve from an initial investment phase to operational sustainability, reaching approaching break-even in Year 4 and achieving stable profitability in Year 5. All financial projections are aligned with the operational strategy and market assumptions described in previous sections, while incorporating conservative estimates and explicit treatment of in-kind contributions, founders' capital, and the risks associated with project funding delays.

Financial Objectives

Over the first five years, NOVA Edge aims to:

- Reach break-even by the fourth year of operation
- Reach a turnover above €700,000 in Year 5
- Maintain a gross margin above 50% from Year 4 onward
- Achieve an EBITDA margin of around 10% by Year 5
- Maintain liquidity above 10% of yearly expenditure
- Demonstrate long-term sustainability through diversified revenue streams
- Reduce operational risk by relying on a mix of R&D project funding, consulting, training and licensing revenues

Cost Structure Assumptions

All personnel costs follow Portuguese labour law, including **14 months of salary per year** and employer social-security contributions (23.75%). CTO and co-CTO contributions appear both as **costs** and **in-kind institutional contributions** (reflecting NOVA FCT's commitment). Personnel accounts for roughly 70% of total expenditure, which is aligned with tech startups.

Personal Costa

The following table presents the plans for hiring and cost of employees. Except for the three junior engineers to be hired at the start of year three, year four, and year five, respectively, all personnel will be hired when the company is created.

For the faculty involved in the project (João Leitão and Nuno Preguiça) we assume that they will dedicate 15% of their time to the management of NOVA Edge, that will be contributed by NOVA FCT in-kind. For completeness the cost of their time is also considered here.

		Gross Anual (€)	Imputed	Additional Taxes (Social Security)	Total Cost for company (€)
CTO	João Leitão	55,302.24	15.00%	23.75%	10,265.48
Co-CTO	Nuno Preguiça	71,641.64	15.00%	23.75%	13,298.48
Product Manager	Diogo Jesus	50,000.00	100.00%	23.75%	61,875.00
Senior Engineer	João Brilha	40,000.00	100.00%	23.75%	49,500.00
Senior Engineer	Rafael Matos	40,000.00	100.00%	23.75%	49,500.00
Communituy Manager	To be hired	28,000.00	100.00%	23.75%	34,650.00
Sales Manager	To be hired	65,000.00	100.00%	23.75%	80,437.50
Junior Engineer (Starting year 3)	To be hired	35,000.00	100.00%	23.75%	43,312.50
Junior Engineer (Starting year 4)	To be hired	35,000.00	100.00%	23.75%	43,312.50
Junior Engineer (Starting year 5)	To be hired	35,000.00	100.00%	23.75%	43,312.50

Below we provide a summary of the personnel costs per year of operation.

	Year 1	Year 2	Year 3	Year 4	Year 5
CTO	10,265.48	10,265.48	10,265.48	10,265.48	10,265.48
Co-CTO	13,298.48	13,298.48	13,298.48	13,298.48	13,298.48
Product Manager	61,875.00	61,875.00	61,875.00	61,875.00	61,875.00
Senior Enginner I	49,500.00	49,500.00	49,500.00	49,500.00	49,500.00
Senior Enginner II	49,500.00	49,500.00	49,500.00	49,500.00	49,500.00
Community Manger	34,650.00	34,650.00	34,650.00	34,650.00	34,650.00
Sales Manager	80,437.50	80,437.50	80,437.50	80,437.50	80,437.50
Junior Enginner I	-	-	43,312.50	43,312.50	43,312.50
Junior Enginner II	-	-	-	43,312.50	43,312.50
Junior Engineer III	-	-	-	-	43,312.50
Total	299,526.46	299,526.46	342,838.96	386,151.46	429,463.96

Operational Costs

Operational costs represent all non-personnel recurring expenses necessary to maintain NOVA Edge's activity. These values remain stable across the first five years, since office space, cloud infrastructure, and support services do not scale significantly with the early team size.

Type of Cost	Annual Cost (€)
Office/lab space	5,000.00
Cloud services, hosting, open-source repositories	4,000.00
Stationary, Printing, Equipment Depreciation	3,000.00
Access to testbed Equipment	10,000.00
Accounting + Auditing Services	6,000.00
Legal Support + IP Protection	10,000.00
Marketing and Dissemination Activities	30,000.00
Total Cost Per Year	68,000.00

Valuation of In-Kind Contribution of NOVA FCT

To ensure the viability of the company, NOVA FCT (major owner of the spin-off) will contribute in-kind. For completeness of this financial report, we quantity the values of those contributions (this is based on estimates considering usual market values for some services). Values reported are per year and we do not expect these values to change significantly during the first five years of operation.

Contribution	Estimated Values (€)
Use of Laboratory and Office space	5,000.00
Access to testbed equipment	10,000.00
IRIS Administrative and Legal Support	16,000.00
Time of CTO (João Leitão) - 15%	10,265.48
Time of Co-CTO (Nuno Preguiça) - 15%	13,298.48
Total In-Kind Contribution per Year	54,563.96

Founder Capital Contribution

Founder	Initial Capital Contribution (€)	Investment Fraction (%)
CTO (João Leitão)	5,000.00	33.33%
Co-CTO (Nuno Preguiça)	5,000.00	33.33%
Product Manager (Diogo Jesus)	2,500.00	16.67%
Senior Engineer (João Brilha)	1,250.00	8.33%
Senior Engineer (Rafael Matos)	1,250.00	8.33%
Total	15,000.00	

At incorporation, NOVA Edge will receive €15,000 in initial paid-in capital from the founding team. These contributions demonstrate commitment to the project. Contributions are as follows: €5,000 from the CTO, €5,000 from the Co-CTO, €2,500 from the Product Manager, and €1,250 each from the two Senior Engineers. This initial capital supports early liquidity and complements institutional in-kind contributions from NOVA FCT.

Revenue Model

Below we describe the forecast for the revenues of NOVA Edge for the first 5 years of operation. In doing this prediction we attempted to be conservative regarding the revenue success of the company.

For this analysis we have considered four main revenue streams:

- Licensing of advanced Babel modules
- Consulting and Integration Services
- Training and Certification Activities
- Research and Development Project Participation

Licensing Revenue Forecast

Here we consider the revenue generated by licensing of Babel specialized modules. We divide this source of revenue in two components, licensing for small and medium enterprises, which we consider as valuable partners for the development of our business model, that we consider 5,000€ per license. We consider also licensing to larger enterprises, here we consider a cost per year of 20,000€.

	Year 1	Year 2	Year 3	Year 4	Year 5
SME Licenses	0	1	2	4	6
Enterprise Licensing	0	0	1	2	3
Licence Revenue over the Year (€)	-	5,000.00	30,000.00	60,000.00	90,000.00

Consulting, Training, and Research Projects Revenue

In the following we make (conservative) projections for the revenue of NOVA Edge in the five first years of operation considering a diversified set of activities. Consulting activities to setup projects for specific clients; training activities to teach how to use the technology for developers; and finally, participation in financed R&D projects. It should be noted that it should be fairly easy to participate in a R&D project in the first year of activity, however, since the calendar for calls (particularly for European projects) might not align with the creation of the startup (and time required to register a PIC) we decided to consider that in the first year of operation NOVA Edge will not be participating in any research and development project.

	Year 1	Year 2	Year 3	Year 4	Year 5
Consulting (€)	30,000.00	60,000.00	80,000.00	120,000.00	150,000.00
Training (€)	10,000.00	20,000.00	25,000.00	35,000.00	40,000.00
R&D Project Participations (€)	-	65,000.00	130,000.00	215,000.00	265,000.00
Total Revenue on Year (€)	40,000.00	145,000.00	235,000.00	370,000.00	455,000.00

Summarization of Revenue

Below we provide a summarization of the revenue considering all revenue sources.

	Year 1	Year 2	Year 3	Year 4	Year 5
Licensing (€)	-	5,000.00	30,000.00	60,000.00	90,000.00
C/T/R&D /€)	40,000.00	145,000.00	235,000.00	370,000.00	455,000.00
Total Revenue (€)	40,000.00	150,000.00	265,000.00	430,000.00	545,000.00

Justifying Forecasted Revenue Streams

The financial plan projects early revenues across four primary streams: licensing, consulting, training, and participation in funded R&D projects. The forecasted values are grounded in realistic industry benchmarks and empirical funding structures.

Licensing Revenue

Annual enterprise license fees of €20,000 and SME license fees of €5,000 are consistent with pricing structures observed in related software sectors. Commercial IoT platforms often charge thousands of euros per year per deployment or per service tier, reflecting the value of operational continuity, device support, and developer tools. For example, professional tiers in well-established IoT platforms are commonly priced between €8,700 and €12,780 per year (or more) depending on the level of included services[1].

These license values position NOVA Edge as a cost-competitive and high-value alternative that emphasizes decentralized operation, offline functionality, privacy, and interoperability, attributes that increasingly differentiate IoT solutions in the market.

Consulting, Training and R&D Contract Revenue

Consulting engagements and training services are priced within typical ranges for specialized IoT software engineering and integration services in Europe, where daily rates often range between €700 and €1,200 per developer or consultant day. These price points reflect established consulting economics for technology delivery and professional education services. Similarly, participation in publicly funded R&D projects, particularly under Horizon Europe and EIC calls, frequently yields €75,000–€250,000 per year depending on project scope, consortium size, and technological ambition; the business plan's

[1] https://www.precedenceresearch.com/iot-platforms-market

assumptions of multiple R&D project contributions per year are thus well founded within observed funding patterns.

Projected Income

In the following we merge the several sources of costs and predicted revenue to analyse the evolution of the project income of NOVA Edge. This analysis allows us to conclude that the company approaches operational break-even in Year 4 and achieves a stable positive operating result in Year 5.

	Year 1	Year 2	Year 3	Year 4	Year 5
Founders Contributions (€)	15,000.00	-	-	-	-
In-Kind Contributions (€)	54,563.96	54,563.96	54,563.96	54,563.96	54,563.96
Personal Costs (€)	299,526.46	299,526.46	342,838.96	386,151.46	429,463.96
Operational Costs (€)	68,000.00	68,000.00	68,000.00	68,000.00	68,000.00
Revenue (€)	40,000.00	150,000.00	265,000.00	430,000.00	545,000.00
EBITDA (€)	(257,962.50)	(162,962.50)	(91,275.00)	30,412.50	102,100.00
EBITDA Margin	-645%	-109%	-34%	7%	19%
Net Result (€)	(257,962.50)	(162,962.50)	(91,275.00)	30,412.50	102,100.00

Cash Flow Analysis

Cash flow analysis is essential for anticipating liquidity needs, mitigating financial risk, and ensuring NOVA Edge maintains operational continuity even under variable revenue conditions. Given the structure of revenues—particularly the dependence on consulting contracts, training cycles, and R&D project reimbursements—cash inflows may be irregular, especially during the early years.

Cash Flow Projections (Years 1–2, Monthly)

The first two years of operation are characterised by negative cash flow, as NOVA Edge invests heavily in core personnel and foundational operational costs before revenues reach a sustainable level. Consulting and training activities start generating income from Year 1, and R&D project participation becomes an important driver of revenue from Year 2 onwards.

In parallel, in-kind contributions from NOVA FCT and an initial founders' capital injection of €15,000 help reduce the effective burden on the company, although they do not remove the need for careful liquidity management.

Key characteristics of Years 1–2 cash flow:

- Average monthly outflow in Year 1 of approximately €27,000 (with peaks in July and November due to double salary payments).
- Average monthly inflow in Year 1 of approximately €3,333.
- Average monthly inflow in Year 2 of approximately €12,500, reflecting the growth of consulting, training, and R&D revenues.
- Liquidity coverage supported by €54,580/year of in-kind contributions from NOVA FCT and €15,000 in founders' capital, which together provide an important buffer but do not fully compensate the negative cash flow in the first two years.
- The main financial risk in this period is the absence of R&D income in Year 1 and the dependency on successfully securing and executing projects in Year 2.

Year 1 - Monthly Cash Flow Overview

- Total annual outflow: €367,526
  - Personnel: €299,526
  - Operational: €68,000 (minus the in-kind contributions)
- Total annual inflow: €55,000
  - Revenue: €40,000
  - Founders' Capital €15,000
- Net annual cash variation: –€257,962.50

Year 1 uses the 14-month salary structure: each employee receives 14 salary instalments per year, with double payments in July and November. This creates two months with significantly higher personnel costs and therefore deeper negative cash flows.

	Founders Capital (€)	Revenue (€)	Personal Costs (€)	Operational Costs (€)	Total Outflow (€)	Net Cash Flow (€)
January	15,000.00	3,333.33	21,394.75	1,119.67	27,061.41	(4,181.08)
February	-	3,333.33	21,394.75	1,119.67	27,061.41	(19,181.08)
March	-	3,333.33	21,394.75	1,119.67	27,061.41	(19,181.08)
April	-	3,333.33	21,394.75	1,119.67	27,061.41	(19,181.08)
May	-	3,333.33	21,394.75	1,119.67	27,061.41	(19,181.08)
June	-	3,333.33	21,394.75	1,119.67	27,061.41	(19,181.08)
July	-	3,333.33	42,789.49	1,119.67	48,456.16	(40,575.83)
August	-	3,333.33	21,394.75	1,119.67	27,061.41	(19,181.08)
September	-	3,333.33	21,394.75	1,119.67	27,061.41	(19,181.08)

October	-	3,333.33	21,394.75	1,119.67	27,061.41	(19,181.08)
November	-	3,333.33	42,789.49	1,119.67	48,456.16	(40,575.83)
December	-	3,333.33	21,394.75	1,119.67	27,061.41	(19,181.08)

Year 2 - Monthly Cash Flow Overview

Year 2 benefits from the start of R&D project participation and growth in consulting and training revenues. However, cash flow remains negative and requires continued external financing.

- Total outflow: €312,962.50
- Total inflow: €150,000
- Net annual cash variation: –€162,962.50

	Founders Capital (€)	Revenue (€)	Personal Costs (€)	Operational Costs (€)	Total Outflow (€)	Net Cash Flow (€)
January	-	12,500.00	21,394.75	1,119.67	22,514.42	(10,014.42)
February	-	12,500.00	21,394.75	1,119.67	22,514.42	(10,014.42)
March	-	12,500.00	21,394.75	1,119.67	22,514.42	(10,014.42)
April	-	12,500.00	21,394.75	1,119.67	22,514.42	(10,014.42)
May	-	12,500.00	21,394.75	1,119.67	22,514.42	(10,014.42)
June	-	12,500.00	21,394.75	1,119.67	22,514.42	(10,014.42)
July	-	12,500.00	42,789.49	1,119.67	43,909.16	(31,409.16)
August	-	12,500.00	21,394.75	1,119.67	22,514.42	(10,014.42)
September	-	12,500.00	21,394.75	1,119.67	22,514.42	(10,014.42)
October	-	12,500.00	21,394.75	1,119.67	22,514.42	(10,014.42)
November	-	12,500.00	42,789.49	1,119.67	43,909.16	(31,409.16)
December	-	12,500.00	21,394.75	1,119.67	22,514.42	(10,014.42)

Year 3 - Quarterly Cash Flow

- Revenue: €265,000 (€66,250 per quarter)
- Personnel costs: €342,838.96 (quarterly distribution accounts for July & November double payments)
- Operational costs: €68,000 minus the in-kind contributions (€3,359.01 per quarter)
- Net cash variation (annual): –€91,275.00

	Founders Capital (€)	Revenue (€)	Personal Costs (€)	Operational Costs (€)	Total Outflow (€)	Net Cash Flow (€)
Q1	-	66,250.00	73,465.49	3,359.01	76,824.50	(10,574.50)
Q2	-	66,250.00	73,465.49	3,359.01	76,824.50	(10,574.50)
Q3	-	66,250.00	97,953.99	3,359.01	101,313.00	(35,063.00)
Q4	-	66,250.00	97,953.99	3,359.01	101,313.00	(35,063.00)

Year 4 - Quarterly Cash Flow

- Revenue: €430,000 (€107,500 per quarter)
- Personnel costs: €386,151.46 (quarterly distribution accounts for July & November double payments)
- Operational costs: €68,000 minus the in-kind contributions (€3,359.01 per quarter)
- Net cash variation (annual): €30,412.50

	Founders Capital (€)	Revenue (€)	Personal Costs (€)	Operational Costs (€)	Total Outflow (€)	Net Cash Flow (€)
Q1	-	107,500.00	82,746.74	3,359.01	86,105.75	21,394.25
Q2	-	107,500.00	82,746.74	3,359.01	86,105.75	21,394.25
Q3	-	107,500.00	110,328.99	3,359.01	113,688.00	(6,188.00)
Q4	-	107,500.00	110,328.99	3,359.01	113,688.00	(6,188.00)

Year 5 - Quarterly Cash Flow

- **Revenue:** €545,000 (€136,250 per quarter)
- **Personnel costs:** €429,463.96 (quarterly distribution accounts for July & November double payments)
- **Operational costs:** €68,000 minus the in-kind contributions (€3.359.01 per quarter)
- **Net cash variation (annual):** +€102,000.00

	Founders Capital (€)	Revenue (€)	Personal Costs (€)	Operational Costs (€)	Total Outflow (€)	Net Cash Flow (€)
Q1	-	136,250.00	92,027.99	3,359.01	95,387.00	40,863.00
Q2	-	136,250.00	92,027.99	3,359.01	95,387.00	40,863.00
Q3	-	136,250.00	122,703.99	3,359.01	126,063.00	10,187.00
Q4	-	136,250.00	122,703.99	3,359.01	126,063.00	10,187.00

Break-Even Analysis

The break-even analysis identifies the moment at which NOVA Edge's operating revenues become sufficient to cover all recurring personnel and operational costs, excluding non-cash in-kind contributions. Based on the current financial projections, the company approaches operational break-even during the fourth year of activity. In that year, revenues reach €430,000, while personnel and operational costs total approximately €386,150 resulting in a small positive margin of €30,412.50. This value indicates that the business becomes structurally sustainable, with all indicators suggesting that even a modest increase in licensing, consulting, or R&D participation would eliminate the remaining gap.

A more granular examination of cash flows reveals that break-even is reached between the first and second quarters of Year 4, when quarterly revenues finally begin to match or exceed quarterly expenditure except in months where double-salary obligations apply. Once Year 4 revenue streams stabilise, the company enters a positive operating cycle, generating the first positive EBITDA during Year 5. Although cumulative liquidity remains negative due to substantial up-front investment and early operating deficits, the business model demonstrates that NOVA Edge becomes operationally viable from Year 4 onward and capable of generating profit in Year 5. This trajectory is typical of research-based deep-tech ventures, where significant early investment precedes revenue acceleration.

Cash Flow Projections for Years 3–5 (Quarterly)

From Year 3 onward, cash-flow projections are computed on a quarterly basis to provide a clear understanding of seasonal financial dynamics, particularly the impact of Portugal's double-salary months and the progressive growth of revenue streams. The company continues to rely on external funding in Year 3, although this year marks the beginning of a more balanced revenue mix incorporating licensing, consulting, training, and participation in multiple R&D projects.

In Year 3, quarterly cash flow remains negative throughout the year, declining from approximately –€530,000 at the end of Year 2 to around –€650,000 at the end of Year 3.". Despite this, Year 3 represents a turning point in terms of revenue maturity, as consulting and R&D income begin to offset a substantial portion of personnel and operational costs. By Year 4, cash outflows decrease relative to inflows as the company enters its break-even region. Although liquidity remains negative, reflecting accumulated deficits from earlier

years, quarterly losses shrink significantly. In Year 4, the company operates with a small positive margin.

In Year 5, NOVA Edge generates a positive annual EBITDA for the first time and achieves a fully profitable operating cycle. Quarterly results alternate between small surpluses in normal-salary months and modest deficits in double-salary periods, but the annual result remains positive. While liquidity does not return to a positive balance within the five-year window due to earlier accumulated deficits, the company reaches a structurally sustainable financial position. It begins the process of rebuilding liquidity reserves, placing itself in a strong position for future expansion, investment attraction, or entry into new markets.

Sensitivity Analysis

A sensitivity analysis was conducted to evaluate the resilience of NOVA Edge's financial model under different risk scenarios, focusing on key variables that could significantly affect revenue and expenditure. Three major scenarios were examined: (i) the failure to secure an expected R&D project, (ii) a six-month delay in revenue from licensing activities, and (iii) a 10% overrun in personnel costs due to labour market pressure.

The first scenario examines the non-award of an R&D project, which would result in a reduction of approximately €75,000 in yearly revenue. This reduction impacts all years from Year 2 onward and would delay operational break-even beyond Year 4. While this represents the most critical single risk factor, the model remains recoverable through practical mitigation strategies such as delaying a junior engineering hire, increasing the number of SME licences, or securing additional national industry-funded innovation contracts. Any of these adjustments would be sufficient to restore the break-even point to Year 4.

The second scenario considers a delay in licensing revenue, assuming licences are secured only in the second half of each year. This scenario has a moderate impact on financial performance, reducing licensing income by 30–50% in affected years. Nevertheless, the company would still be able to maintain the operational break-even point in Year 4 due to the strength and reliability of consulting and R&D revenue streams. Small adjustments in service pricing or acquisition of additional consulting engagements would further stabilise this trajectory.

The third scenario evaluates a 10% increase in personnel costs, which would raise annual expenditure by €30,000–€40,000. This situation would delay operational break-even by approximately one year. However, the model demonstrates that deferring a single junior hire or increasing the number of R&D projects can neutralise the impact entirely. Given the high

likelihood of securing multiple R&D projects under Horizon Europe and national funding schemes, NOVA Edge retains robust capacity to absorb such fluctuations.

Taken together, these scenarios demonstrate that NOVA Edge's financial model is resilient and adaptable. While early years rely heavily on external funding and controlled expenditure, the combination of diversified revenue streams, conservative cost assumptions, and flexibility in recruitment ensures the company's ability to maintain a break-even trajectory and achieve long-term financial sustainability.

Ensuring Sustainability and Liquidity for NOVA Edge

The financial plan presented in this document shows that with a small investment NOVA Edge can become a sustainable company by year four of operation, however it is essential to ensure the investment required for the three first years of operation. While a possibility would be to attract external investment from a VC company in exchange for equity, a more interesting approach is to take advantage of the strong roots of NOVA Edge on research and innovation to pursue the required funding through research and innovation programmes at the European level.

To that end, NOVA Edge will actively pursue European (and potentially Portuguese national) research funding programmes that support technological maturation, business validation, and interoperability testing, particularly in the domains of decentralized systems, IoT and domotics, and edge AI.

The European Union's Horizon Europe programme is the primary framework for research and innovation support through 2027, with a total indicative budget of EUR 93.5 billion aimed at facilitating collaborative technology development across key digital sectors. This programme includes open calls addressing AI, IoT, edge computing, and digital infrastructure research, providing a well-structured pathway for deep-tech startups to obtain funding.

Within Horizon Europe, the European Innovation Council (EIC) is a dedicated instrument to accelerate innovations with strong commercial potential. In its 2025 work programme, the EIC earmarked over €1.4 billion to support breakthrough technologies, offering substantial grants for innovation projects and scaling activities in strategic digital fields.

Specifically, the EIC Transition funding line provides grants ranging from €0.5 million to €2.5 million to validate and demonstrate technology in real-world environments and establish a market-ready business case. Such funding levels are materially significant relative to NOVA Edge's projected operating expenses in the first three years (approx. €600k–€700k) and could therefore sustain core operations if successful applications are awarded. Participation in

these calls also positions the company within a broader network of European innovators and potential industry partners, which can play a relevant role in accelerating the revenue streams of NOVA Edge.

In addition to direct EU funding, NOVA Edge can leverage cascade funding mechanisms (also known as Financial Support to Third Parties) where open calls under larger EU projects provide SME-focused grants of up to €300,000 per project, facilitating technology validation and early adoption without complex and time-consuming consortium obligations.

This research funding strategy will reduce reliance on early commercial revenue and provide a financially sound transition path from prototype validation to market readiness while controlling personnel and operating costs.