

Fachhochschule Dortmund
University of Applied Sciences and Arts
Embedded Systems Engineering

A Comparative Analysis of System Architectures for Collaborative SLAM

Research Thesis

**Fachhochschule
Dortmund**

University of Applied Sciences and Arts

Author: Aditya Kumar
Matriculation Number: 7219675
Supervisor: Prof. Dr. Stefan Henkler
Co-Supervisor: -
Date: February 6, 2026

Abstract

Collaborative Simultaneous Localisation and Mapping (C-SLAM) enables teams of robots to construct a shared environment map while estimating their trajectories. As multi-robot systems move toward larger deployments in dynamic settings, the backend architecture becomes a primary determinant of end-to-end performance. However, experimentally controlled comparisons between different C-SLAM architectures with similar frontend-backend processing under realistic network effects remain scarce.

This research thesis presents a systematic, implementation-grounded comparison of two representative backend paradigms within a unified evaluation framework: a centralised incremental pose graph optimiser and a decentralised DDF-style distributed optimiser. To ensure architectural fairness, both backends consume identical pre-computed factor streams from the COSMO-Bench dataset and are evaluated using a fully automated ROS 2 testbed that applies packet loss, transmission delay, and bandwidth limitations in a controlled manner.

Performance is quantified using localisation accuracy, latency and convergence behaviour, communication bandwidth, and computational resource utilisation. The centralised backend achieves superior global consistency (ATE RMSE 1.89–2.53 m across evaluated scenarios) and rapid team-level convergence ($T(team)_{conv} < 0.2s$) as team size grows from three to five robots. However, it incurs asymmetric communication patterns with downlink traffic comprising 80–90% of total bandwidth (26–49 MiB per run) and concentrates processing at the server.

The decentralised backend demonstrates regime-dependent performance characteristics. Under ProRadio networking, it maintains competitive accuracy (ATE RMSE 2.72–3.82 m) with peer-to-peer traffic 12–45% lower than centralised uplink requirements. Under WiFi networking, accuracy degrades at larger team sizes (ATE RMSE 6.66 m at five robots) despite increased interface coordination activity. Both architectures exhibit sensitivity to severe bandwidth constraints (0.25 Mbps caps increase dissemination latency 8–10 \times for centralised; interface stabilisation becomes inconsistent for decentralised), but the decentralised implementation encounters stability failures under intermittent connectivity that limit robustness claims in the current implementation.

Overall, neither centralised nor decentralised C-SLAM paradigms are universally optimal; their effectiveness depends on specific deployment requirements. Centralised architectures are preferable when stable infrastructure is available, global consistency is critical, and team sizes remain small to medium (≤ 5 robots in the evaluated scenarios). Decentralised architectures are appropriate when bandwidth is severely constrained, peer-to-peer communication is reliable, and the constraint regime supports effective distributed coordination—though practical deployment requires resolving the identified software stability issues under network intermittency.

Declaration

I, Aditya Kumar, hereby confirm, that I have written the Research Thesis at hand independently – in case of a group work: my respectively designated part of the work -, that I have not used any sources or materials other than those stated, and that I have highlighted any citations properly.
Dortmund, February 6, 2026

Aditya Kumar

Contents

1	Introduction	1
1.1	Motivation	1
1.2	Problem statement	2
1.3	Research Goals	3
1.4	Structure of the Thesis	3
2	Fundamentals of SLAM and C-SLAM	4
2.1	Single-Agent SLAM	4
2.1.1	Problem Definition	5
2.1.2	Graph-Based SLAM Framework	5
2.1.3	SLAM System Architecture	8
2.2	C-SLAM	9
2.2.1	Multi-Robot State and Pose Graph	10
2.2.2	Inter-Robot Loop Closures	10
2.2.3	Architectural Classes for C-SLAM	10
3	State of the Art	13
3.1	Centralised C-SLAM	13

3.1.1	Architectural Patterns	13
3.1.2	Performance Implications	14
3.2	Decentralised/Distributed C-SLAM	15
3.2.1	Architectural Patterns	15
3.2.2	Performance Implications	16
3.3	Comparative Analysis	17
4	Methodology	19
4.1	Experimental Design	19
4.2	State Representation and Estimation Variables	20
4.2.1	Pose Variables	21
4.2.2	Multi-Robot Global State	21
4.2.3	Constraint Sets and Pose Graph	21
4.2.4	Reference Optimisation Problem	22
4.3	C-SLAM Backend Architectures	22
4.3.1	Centralised C-SLAM Backend	22
4.3.2	Decentralised Distributed SLAM Backend	23
4.4	Communication Model and Network Variables	24
4.4.1	Communication Model	24
4.4.2	Network Impairment Model	24
4.5	Performance metrics	25
4.5.1	Timing metrics	25
4.5.2	Accuracy Metrics	27
4.5.3	Resource Metrics	27
5	Implementation	29
5.1	Dataset and Frontend	29
5.2	System Overview	30

5.3	Operationalisation of the Experimental Design Space	30
5.3.1	ROS 2 Factor Streaming	31
5.4	Centralised Backend	32
5.4.1	Factor Ingestion and Batching	33
5.4.2	Solver: Incremental Optimisation with iSAM2	34
5.4.3	Optional Downlink and ACK Instrumentation	34
5.5	Decentralised Backend	35
5.5.1	Deterministic Factor Partitioning	35
5.5.2	Solver: Local Optimisation and Remote-Prior Refresh	36
5.5.3	Interface Exchange and Coordination Loop	37
5.5.4	Communication Transport and Message Schema	37
5.6	Network Impairment and Quality-of-Service Implementation	38
5.6.1	Impairment Injection Points	38
5.6.2	Impairment Semantics Relevant to KPIs	38
5.6.3	Impairment Configuration	39
5.6.4	Quality-of-Service Configuration	39
5.6.5	Team Size Scaling and Namespace Isolation	39
5.7	Performance Metrics Instrumentation	40
5.7.1	Event Logging and Derivation Pipeline	40
5.7.2	Latency Instrumentation	40
5.7.3	Accuracy Instrumentation	40
5.7.4	Resource and Bandwidth Instrumentation	41
6	Evaluation	42
6.1	Experimental Protocol	42
6.1.1	Timing Semantics	43
6.2	Results Overview	43
6.3	Baseline Comparison Under Native Dataset Networking	44

6.3.1	Reference r3 Baseline: Centralised vs. Decentralised . . .	44
6.3.2	Scalability Trend (r3–r5)	46
6.4	Sensitivity to ROS 2 QoS Policy	49
6.5	Sensitivity to Synthetic Network Impairments	51
6.5.1	Centralised Backend Under Impairments	51
6.5.2	Decentralised Backend Under Impairments	52
6.6	Robustness and Unsupported Configurations	53
6.6.1	Failure Mode Analysis (decentralised crashes)	54
6.7	Synthesis and Limitations	55
7	Conclusion and Future Works	57
A	Appendix	A1
A.1	Supplementary Evaluation Material	A1
A.1.1	Full baseline scalability tables (split)	A1
A.1.2	Decentralised diagnostic counts	A3
A.1.3	Unsupported configurations	A3
A.1.4	Diagnostic figures	A3

List of Figures

2.1	Bipartite factor-graph representation of SLAM: circular pose nodes \mathbf{x}_t , rectangular landmark nodes \mathbf{l}_i , and square factor nodes (black) encoding constraints through different edge types (odometry, landmark observations, loop closures, prior).	6
2.2	Generic SLAM pipeline with frontend and backend.	8
2.3	C-SLAM architectures: centralised, decentralised, and hybrid/hierarchical.	11
5.1	Sequence diagram depicting agent–central server interaction. . .	33
5.2	Sequence diagram depicting decentralised agent interaction and interface exchange.	35
6.1	Efficiency-normalised metric for the r3 baseline: CPU-seconds per optimisation step. Bars show mean over repeats ($R = 5$); error bars denote 95% CI across repeats, with individual repeats overlaid. Lower values indicate less compute time spent per solver update.	45

6.2	Compact summary of the r3 baseline (Table 6.1): ATE RMSE and total traffic with uplink/downlink split. Bars show mean over repeats ($R = 5$); error bars denote 95% CI across repeats.	46
6.3	Scalability trend in the baseline sweep (Table 6.2): ATE RMSE and traffic vs. team size (r3–r5) for WiFi and ProRadio. Lines show mean over repeats ($R = 5$); error bars denote 95% CI across repeats.	48
6.4	Pareto view of baseline trade-offs across team sizes (r3–r5): ATE RMSE vs. total traffic for each successful repeat. This visualisation complements Figure 6.3 by showing per-run dispersion and the accuracy–bandwidth frontier for each architecture.	48
6.5	Decentralised diagnostic: ATE mean vs. DDF termination time. The x-axis reports $T_{\text{stop}}^{(\text{team})}$ (time from <code>input_end</code> until all agents emit <code>ddf_stop</code> , team statistic = max over robots). Error bars denote 95% CI over repeats ($R = 5$).	49
6.6	QoS sensitivity (r3): summary of how ATE and traffic shift from the baseline profile (RV-20) under BT-TL-10 and BT-TL-50. Lines show mean over repeats ($R = 5$); error bars denote 95% CI across repeats. Failed configurations are marked.	50
6.7	Normalised impairment sensitivity (slowdown vs. baseline) under uplink-only (publisher-impaired) degradations (Tables 6.4 and 6.5): centralised $t_{\text{global}}/t_{\text{base}}$ and decentralised $\text{Iface p95}/t_{\text{base}}$. Points show mean over repeats ($R = 5$); error bars denote 95% CI across repeats. Failed runs are marked.	53

6.8	Robustness matrix summarising which configurations are supported (all repeats ok) vs. unsupported (at least one repeat fails) across the three studies: baseline, QoS, and impairments. Green ok indicates all repeats completed successfully; red fail indicates at least one repeat failed; grey n/a indicates a configuration not evaluated in this study (QoS is evaluated at r3 only; impairments at r3 and r5 only).	54
A.1	Resource traces over time for the r3 baseline: centralised CPU/RSS of the server process vs. decentralised CPU/RSS aggregated across agents (sum over per-agent processes). Time is aligned to the start of each run. These traces support diagnosis of stability issues (e.g., monotonic RSS growth preceding agent crashes) and complement the aggregate means reported in Table 6.1.	A4
A.2	Absolute timing under impairments (supplementary): centralised t_{global} and decentralised interface stabilisation latency across impairment conditions. This figure complements the normalised slowdown view in Figure 6.7.	A4

List of Tables

3.1	Comparison of centralised and decentralised C-SLAM architectures. These trends are consistent across visual, LiDAR, and multi-modal systems.	17
5.1	Repository layout of the experimental framework.	31
5.2	Operationalisation of the design space into concrete runtime knobs.	32
6.1	Baseline comparison for r3 under dataset-native networking. t_{global} is a centralised dissemination proxy (time until last global map broadcast). “Corr. p95” is a correction-latency indicator. For centralised runs it reports loop-closure correction latency (p95), while for decentralised runs it reports interface correction stabilisation latency (p95). These timing indicators are architecture-specific. Although Strict cross-architecture convergence comparisons should use the shared stabilisation proxy $T_{\text{conv}}^{(\text{team})}$ (Section 6.1.1). Link(L): ProRadio(P) & WiFi (W); Backend(B): Centralised(C) & Decentralised(D).	45

6.2	Condensed baseline scalability trend across team sizes (r3–r5). Link: P=ProRadio, W=WiFi. Backend: C=centralised, D=decentralised. “Time” denotes t_{global} for centralised runs (dissemination proxy) and Corr. p95 for decentralised runs (correction-propagation proxy). $T_{\text{conv}}^{(\text{team})}$ is the shared stabilisation proxy (mean \pm 95% CI; “ \geq ” indicates right-censoring when stabilisation is not observed).	47
6.3	QoS sensitivity at r3. Two alternate profiles are evaluated: BT-TL- 10 and BT-TL-50, which use best-effort reliability and transient- local durability with history depth 10 and 50, respectively. Base- line (RV-20) results are reported in Table 6.1. “Time” denotes t_{global} for centralised runs and Corr. p95 for decentralised runs. $T_{\text{conv}}^{(\text{team})}$ is the shared stabilisation proxy (mean \pm 95% CI; “ \geq ” indicates right-censoring). Rows marked fail indicate at least one repeat crashed. Link(L) = WiFi(W) & ProRadio(P); QoS profile = BT-TL-10/50	50
6.4	Impact of representative network impairments on the centralised backend. Bandwidth caps of 1–3 Mbps were near-identical to baseline and are omitted here; only the most restrictive cap (0.25 Mbps) is shown. Wr3/5 = WiFi r3/5; Pr3/5 = ProRadio r3/5; bwcap = bwcap 0.25 Mbps; FDM = Factor Delivery Time .	51
6.5	Impact of representative network impairments on the decentralised backend (DDF-style). “Iface p95” is interface correction stabilisa- tion latency (p95). Rows marked fail indicate runs where at least one agent crashed. Wr3/5 = WiFi r3/5; Pr3/5 = ProRadio r3/5; bwcap = bwcap 0.25 Mbps; FDM = Factor Delivery Time	52

A.1	Full baseline scalability trend across team sizes (r3–r5), results view. Link: P=ProRadio, W=WiFi. Backend: C=centralised, D=decentralised. “Time” denotes t_{global} for centralised runs (dissemination proxy) and Corr. p95 for decentralised runs (correction-propagation proxy). The shared convergence proxy $T_{\text{conv}}^{(\text{team})}$ is reported as the comparison anchor (mean \pm 95% CI across repeats; “ \geq ” indicates right-censoring when stabilisation is not observed).	A2
A.2	Full baseline scalability trend across team sizes (r3–r5), systems view. Opt. p95 is the p95 optimisation-step duration. CPU and RSS are run means (server for centralised; sum over agents for decentralised).	A2
A.3	Decentralised diagnostic counts in the baseline sweep: number of inter-robot loop-closure constraints (Between factors whose endpoint keys belong to different robot symbols) and number of interface correction updates (count of stabilised interface-correction events). Values are mean \pm 95% CI over repeats ($R = 5$).	A3
A.4	Runs that did not complete successfully and were excluded from quantitative comparisons.	A3

Introduction

1 Motivation

In recent years, a sharp increase in the use of autonomous systems in a multitude of domains has driven significant advances in the field of robotics. These systems have been used in various industries, such as healthcare [27], transportation [26], and manufacturing [15], to improve efficiency, precision, and overall performance.

In this increasingly interconnected and complex world, the limitations of single-agent, centralised, monolithic systems are becoming more apparent. As a result, there is a growing trend towards the development of multi-agent systems that can work collaboratively and adaptively to tackle complex problems. These systems are able to distribute decision-making and problem-solving processes across multiple agents, allowing greater flexibility, scalability, and efficiency in the face of dynamic environments. Multi-agent systems are often chosen for problems that are inherently distributed, large, and complex for a single agent to solve. Examples include search and rescue operations [9], environmental monitoring [13], precision agriculture [23] (e.g. monitoring crop health and controlling pest infestations), and mapping of archaeological sites [20].

While multi-agent systems are distributed in their true nature, the design and architecture for coordination and communication between multiple agents must be carefully planned to ensure seamless collaboration and task completion. This involves developing algorithms that enable agents to share information, make decisions collectively, and adapt to changing environments in real time.

A fundamental requirement for these coordinated tasks is a shared, real-time understanding of the environment, often represented as a global map. This is especially critical in scenarios where external positioning systems such as GPS are inconsistent or entirely unavailable, necessitating a self-contained perception solution. For a traditional single-agent system, this is solved by Simultaneous Localisation and Mapping (*SLAM*) [29], which allows the agent to build a map of its environment while simultaneously localising itself within that map. In a multi-agent setting, this extends to collaborative SLAM. *C-SLAM* leverages multiple robots working together to overcome the limitations of single-agent systems, such as limited perspectives, error accumulation over time, and challenges of large-scale complex environments. By sharing sensory information and coordinating efforts, multi-agent teams can create more accurate and robust maps while also increasing operational efficiency [35].

2 Problem statement

While many factors contribute to successful development and deployment of a C-SLAM system, architecture plays a vital role. Architectural choices determine how sensing, communication, and optimisation workloads are distributed across robots and infrastructure, and directly affect scalability, robustness, and real-time performance. Key architectural paradigms, ranging from *purely centralised* setup where all map building and optimisation tasks are offloaded on a centralised server to *decentralised* where each robot maintains a local estimate and global consistency is recovered through distributed coordination. Each paradigm introduces inherent trade-offs in estimation accuracy, communication bandwidth requirements, computational load, and tolerance to network impairments.

Architectural selection in C-SLAM systems often relies on intuition, with decentralised architectures recognised for their robustness and scalability due to eliminating single points of failure. However, they may incur higher latency from increased coordination needs. In contrast, centralised pipelines offer faster global consistency but can become bottlenecks as team sizes grow or network conditions worsen. Despite these widely held assumptions and a substantial body of existing research on C-SLAM architectures and optimisation techniques, most studies evaluate systems under idealised or loosely controlled communication conditions, or focus on a single architectural paradigm in isolation. As a result, there is a lack of systematic, controlled, and reproducible comparative assessment that isolates architectural effects, characterises latency and convergence under increasing collaboration scale, and relates performance to computational and communication constraints. Addressing this gap is necessary to enable principled architectural selection rather than assumption-driven design.

3 Research Goals

The primary objective of this research project is to conduct and establish an evidence-based understanding of how architectural choices in C-SLAM shape the core trade-offs between *global consistency*, *robot autonomy*, and system performance. To support principled architecture selection, this work aims to quantify and relate key characteristics such as latency, scalability limits, robustness, and implementation complexity. The key research questions to be addressed are:

1. How do architectures differ in terms of global map update latency and convergence time?
2. What is the impact on latency as the number of collaborating agents increases?
3. What are the computational loads and communication bandwidth requirements of each architecture under varying network constraints, and how do these evolve under constrained network conditions?
4. How does the choice of architecture affect the system’s robustness to common real-world challenges?

4 Structure of the Thesis

This thesis is organised into six chapters. Chapter 2 introduces the fundamental concepts of graph-based SLAM and C-SLAM, establishing the theoretical basis for the subsequent analysis. Chapter 3 reviews the state of the art in collaborative SLAM and builds a comparative analysis grounded in existing literature. Chapter 4 outlines the structure of the comparative framework and the design of the simulation-based evaluation strategy. Chapter 5 details the implementation of the experimental testbed, including the deployment of representative collaborative SLAM architectures. Chapter 6 presents and analyses the comparative results with respect to performance, latency, resource utilisation, and robustness. Finally, Chapter 7 summarises the key findings, draws conclusions, and discusses directions for future research.

Fundamentals of SLAM and C-SLAM

1 Single-Agent SLAM

Simultaneous Localisation and Mapping (SLAM) is a state estimation problem in which an agent or a mobile robot is tasked to build a map of an unknown environment while simultaneously localising itself within that map based on on-board sensor measurements [12, 1]. At the start of a mission, the robot has no prior knowledge of its environment, and its initial pose defines the coordinate frame of reference. As the robot moves, it measures and maps the environment with reference to this initial position, therefore making the process of mapping incremental in nature. However, inaccuracies in the robot's movement lead to an increasing deviation from the true position, potentially degrading the map quality. To counteract this, the robot needs to consistently optimise its estimated trajectory and map representation through techniques such as sensor fusion, loop closure detection, and nonlinear optimisation, thereby minimising accumulated error and improving overall accuracy.

Modern SLAM systems address this challenge by fusing heterogeneous sensor data (e.g., wheel odometry, inertial measurements, cameras, and LiDAR) into a consistent probabilistic estimate of the robot's pose and the surrounding environment. A central idea is to formulate SLAM as an optimisation problem over a set of latent variables (poses and possibly landmarks) that best explain

the observed measurements. This section introduces the standard graph-based formulation that underpins the rest of this research thesis and briefly outlines the typical system architecture of a single-agent SLAM pipeline.

1.1 Problem Definition

Let $x_t \in SE(d)$ denote the robot pose at discrete time (or keyframe) index t , where $d \in \{2, 3\}$ is the operational dimension and $SE(d)$ is the special Euclidean group. We write a pose as $x_t = (t_t, r_t)$ with translation $t_t \in \mathbb{R}^d$ and orientation $r_t \in SO(d)$, where

$$SO(d) \triangleq \{R \in \mathbb{R}^{d \times d} \mid R^\top R = I, \det(R) = 1\}. \quad (2.1)$$

Let m denote a map of the environment (e.g., a set of landmarks or a continuous representation). Over a horizon $t = 0, \dots, T$, the robot receives control inputs $u_{0:T-1}$ and sensor measurements $z_{0:T}$. The probabilistic SLAM problem is to estimate the joint posterior

$$p(x_{0:T}, m \mid z_{0:T}, u_{0:T-1}), \quad (2.2)$$

where $x_{0:T} = \{x_0, \dots, x_T\}$ denotes the trajectory.

In practice, SLAM backends typically seek a maximum a posteriori (MAP) estimate under modelling assumptions (motion and measurement models, priors, and conditional independence), which yields a sparse optimisation problem (Section 2.1.2). While Eq. (2.2) includes an explicit map variable m , this thesis focuses on *pose graph* formulations where the environment is represented implicitly through relative constraints and the backend optimises over poses only. This choice matches the backend-centric scope of the thesis and the experimental setup introduced in Chapter 4.

1.2 Graph-Based SLAM Framework

Graph-based SLAM represents the estimation problem as a sparse graphical model where nodes correspond to unknown variables (robot poses and, optionally, landmarks), and edges correspond to probabilistic constraints derived from sensor measurements. In the SLAM literature, such edges are commonly referred to as *factors* [12]. Throughout this thesis, the terms *factor* and *constraint* are used interchangeably, both denote a measurement-derived relationship between a subset of state variables.

In the remainder, discrete keyframes/poses are indexed by i, j , and reserve t for time.

Constraints as Pairwise Factors

The pose graph setting used here is naturally expressed in terms of pairwise constraints between poses. Let \mathcal{Z} denote the set of relative-pose measurements and let \mathcal{E} denote the set of constrained pose index pairs (i, j) . Each measurement

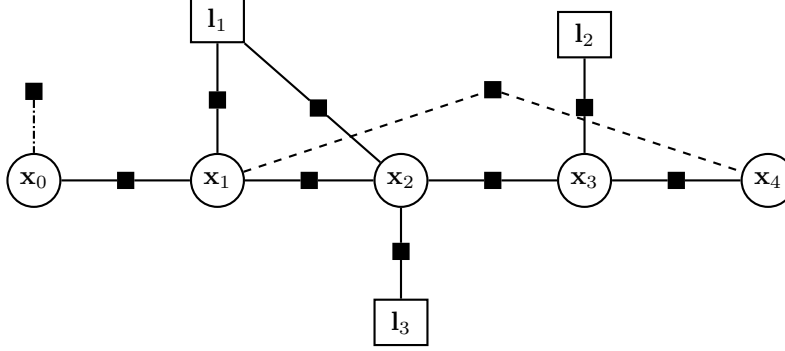


Figure 2.1: Bipartite factor-graph representation of SLAM: circular pose nodes \mathbf{x}_t , rectangular landmark nodes \mathbf{l}_i , and square factor nodes (black) encoding constraints through different edge types (odometry, landmark observations, loop closures, prior).

$z_{ij} \in \mathcal{Z}$ induces a likelihood term (factor) connecting two pose variables x_i and x_j :

$$f_{ij}(x_i, x_j) \propto \exp\left(-\frac{1}{2} \|z_{ij} - h(x_i, x_j)\|_{\Omega_{ij}}^2\right), \quad (2.3)$$

where $h(x_i, x_j)$ predicts the relative transformation implied by the current pose estimates and Ω_{ij} is the information matrix (inverse covariance) weighting the residual. The Mahalanobis norm is

$$\|e\|_{\Omega_{ij}}^2 \triangleq e^\top \Omega_{ij} e. \quad (2.4)$$

Conceptually, z_{ij} encodes “what the sensors say” about the relative pose between two states (e.g. odometry between consecutive keyframes, or a loop closure between revisited places). The function $h(\cdot)$ encodes “what the current state estimate predicts” for the same relative relation.

Typical constraint types in pose graph SLAM include:

- **Motion constraints** between consecutive keyframes (short-horizon odometry / scan matching / IMU-derived relations),
- **Intra-robot loop closures** between non-consecutive poses when the robot revisits a place, and
- **Prior constraints** that anchor otherwise unobservable degrees of freedom (to resolve gauge freedom).

Nonlinear Least-Squares Optimisation Template

Under standard conditional-independence assumptions, the posterior over the unknown variables can be factorised into a product of likelihood and prior terms,

which is naturally represented as a factor graph (Section 2.1.2). For a generic set of factors $\{\phi_k\}$, each involving a subset of variables $X_k \subseteq X$, we write

$$p(X \mid \mathcal{Z}) \propto \prod_k \phi_k(X_k). \quad (2.5)$$

Assuming independent, zero-mean Gaussian noise models, each factor can be expressed in exponential form and MAP estimation reduces to a nonlinear least-squares (NLLS) problem:

$$X^* = \arg \min_X \sum_k \|e_k(X_k)\|_{\Omega_k}^2, \quad (2.6)$$

where $e_k(\cdot)$ is the residual induced by factor k and Ω_k is the corresponding information matrix.

A particularly important special case for this thesis is *pose graph SLAM*, in which all factors are (typically) pairwise relative-pose constraints between pose variables. Let \mathcal{E} denote the set of constrained index pairs (i, j) and let z_{ij} be the corresponding relative measurement. The canonical pose graph objective becomes

$$X^* = \arg \min_X \sum_{(i,j) \in \mathcal{E}} \|z_{ij} - h(x_i, x_j)\|_{\Omega_{ij}}^2. \quad (2.7)$$

Equation (2.7) is the *reference optimisation template* used throughout this thesis. Later chapters specialise it to multi-robot pose graphs, and the implementation chapters show how different backends solve the same objective under different communication and computation constraints.

Solving the Optimisation Problem

The objective in Eq. (2.7) is nonlinear because poses live on a manifold and relative-pose prediction involves nonlinear composition. Backends therefore solve it iteratively by linearising residuals around a current estimate and computing an incremental update.

Let δ denote a local perturbation in a tangent space and let \oplus denote a retraction operator that maps a perturbation back to the manifold. Linearising a residual term yields

$$(z_{ij} - h((x_i \oplus \delta_i), (x_j \oplus \delta_j))) \approx (z_{ij} - h(x_i, x_j)) + J_{ij} \delta, \quad (2.8)$$

with Jacobian J_{ij} evaluated at the current estimate. Stacking all terms produces a sparse linear least-squares problem in δ , which can be solved efficiently using sparse factorisation. In practice, Gauss–Newton or Levenberg–Marquardt variants are used.

Two solver styles are common:

- **Batch optimisation**, which periodically relinearises and solves the full problem, and

- **Incremental optimisation**, which updates the solution efficiently as new factors arrive.

Incremental smoothing and mapping methods (e.g. iSAM2) exploit the sparsity structure to update factorisations efficiently and are well suited to the streaming factor setting used in this thesis (see Chapter 5, centralised backend).

1.3 SLAM System Architecture

While the graph-based formulation defines the underlying estimation problem, practical SLAM systems are organised as modular pipelines that process sensor data in real time. A widely adopted abstraction is the division into a *frontend*, which processes raw sensor data and proposes constraints, and a *backend*, which maintains and optimises the factor graph [12]. Figure 2.2 conceptually illustrates this split.

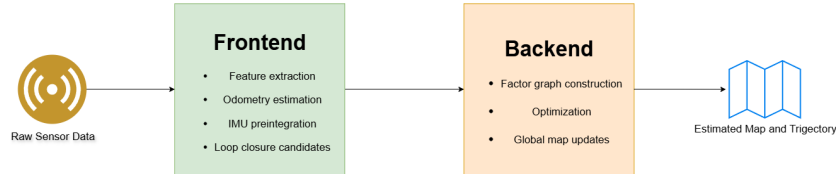


Figure 2.2: Generic SLAM pipeline with frontend and backend.

Frontend

The frontend converts raw sensor streams into a compact set of geometric and topological constraints. Its typical responsibilities include:

- **Preprocessing and calibration:** Synchronisation of multi-sensor data, undistortion of images, and application of extrinsic and intrinsic calibration.
- **Motion estimation:** Computation of relative pose estimates between consecutive frames or scans using visual odometry, LiDAR odometry, or IMU preintegration.
- **Feature extraction and matching:** Detection and description of salient features (e.g. keypoints, edges) and data association across observations.
- **Keyframe selection:** Sub-sampling of frames or scans to maintain a tractable problem size while preserving geometric coverage.
- **Outlier rejection and place recognition:** Application of geometric consistency tests (e.g. RANSAC, epipolar constraints) and appearance-based retrieval (e.g. bag-of-words, deep embeddings) to filter spurious matches and propose candidate loop closures.

The primary output of the frontend is a set of candidate factors that can be added to the factor graph for the backend optimisation. The frontend does not guarantee long-term global consistency and its estimates are prone to drift.

Backend

The backend maintains the current factor graph and computes the robot trajectory and map estimates by solving Eq. (2.7). Its responsibilities typically include:

- **Graph management:** Inserting new nodes and factors as they are proposed by the frontend, and optionally marginalising or pruning variables to control computational complexity.
- **Optimisation scheduling:** Deciding when to trigger incremental updates versus more expensive batch re-optimisations, trading off accuracy and latency.
- **Linearisation and solving:** Building and solving the linearised normal equations using sparse matrix factorisation or incremental factorisation updates.
- **State dissemination:** Providing updated pose and map estimates to downstream components such as control, planning, or visualisation.

Intra-Robot Loop Closure

As the robot explores, its trajectory may revisit previously mapped areas. Detecting such events is known as *loop closure detection*. A constraint is added between non-consecutive poses (i, j) . In the single-robot setting, such a constraint is called an *intra-robot loop closure*. Loop closures are crucial as they introduce long-range information that corrects accumulated drift and improves global consistency.

2 C-SLAM

Collaborative SLAM (*C-SLAM*) generalises SLAM to a team of robots that exchange information to jointly estimate their trajectories and (implicitly or explicitly) a shared map [1, 6]. From an estimation perspective, the extension is rather simple, the variable set grows from one trajectory to multiple trajectories, and additional inter-robot constraints couple those trajectories but from a systems perspective, the extension becomes tedious as communication bandwidth, latency, temporary partitions, and compute limitations constrain how the global optimisation problem can be represented and solved in practice.

2.1 Multi-Robot State and Pose Graph

Consider a team of N robots. For each robot $r \in \{1, \dots, N\}$, let T_r denote the index set of keyframes (pose nodes) associated with that robot, and let $x_i^{(r)}$ denote the pose of robot r at keyframe $i \in T_r$. The global multi-robot state is the set of all poses across all robots:

$$X = \left\{ x_i^{(r)} \mid r \in \{1, \dots, N\}, i \in T_r \right\}. \quad (2.9)$$

The measurement set \mathcal{Z} contains relative pose constraints within each robot and across robots. The corresponding edge set \mathcal{E} can be conceptually decomposed into:

- **Local edges** (within a robot): odometry and intra-robot loop closures, and
- **Inter-robot edges** (across robots): constraints connecting $x_i^{(r)}$ and $x_j^{(s)}$ with $r \neq s$.

while the optimisation objective remains the same as in the single-robot case that is to estimate X so that all relative constraints are satisfied as well as possible. The difference is that the graph now consists of multiple per-robot subgraphs connected by a set of inter-robot edges.

2.2 Inter-Robot Loop Closures

In addition to intra-robot loop closures, multi-robot systems form *inter-robot loop closures* when different robots observe common structure or each other. These events introduce constraints between poses belonging to different robots, i.e. factors connecting $(x_i^{(r)}, x_j^{(s)})$ with $r \neq s$. Inter-robot loop closures are particularly valuable because they align local frames and correct drift between agents, enabling a globally consistent multi-robot estimate.

2.3 Architectural Classes for C-SLAM

The factor-graph formulation above describes a conceptually centralised optimisation problem over all robots and landmarks. However, implementing a fully centralised solution is often impractical due to communication bandwidth limits, latency constraints, and the desire for robustness to node failures. Consequently, a wide range of C-SLAM architectures has been proposed, which can be broadly classified into *centralised*, *decentralised*, and *hybrid or hierarchical* designs [1, 6]. Figure 2.3 illustrates these high-level classes.

Centralised Architectures

In a centralised C-SLAM architecture, a dedicated server (or backend node) collects information from all robots and performs the global optimisation. Each

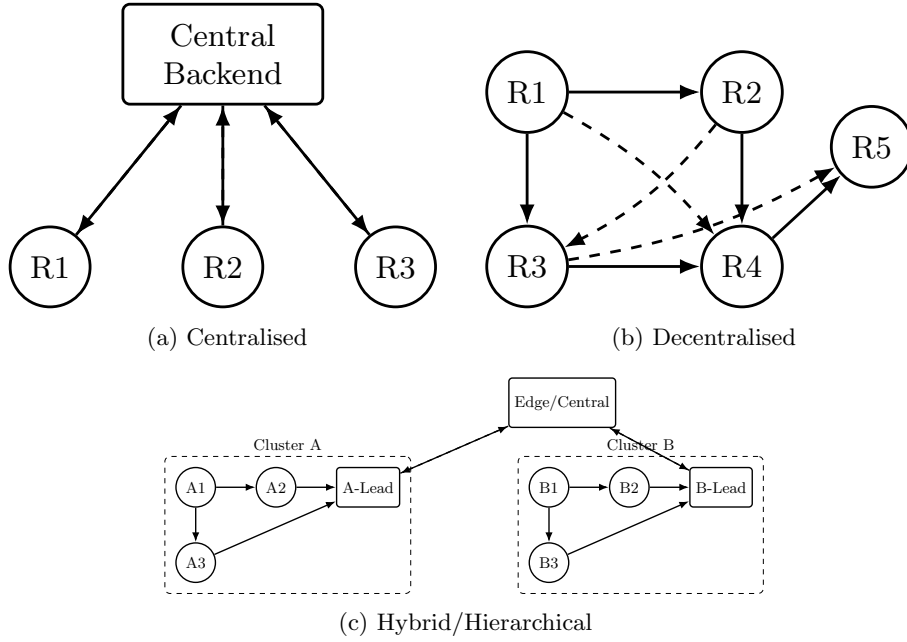


Figure 2.3: C-SLAM architectures: centralised, decentralised, and hybrid/hierarchical.

robot typically runs a lightweight local frontend that estimates its own motion and detects intra-robot loop closures. The robots then transmit summarised information (e.g. pose graph constraints, keyframes, or local maps) to the central server, which maintains a global factor graph and periodically computes a joint estimate of all robot trajectories and the environment.

Decentralised Architectures

Decentralised C-SLAM architectures avoid a dedicated central server. Instead, each robot maintains its own local factor graph and collaborates with other robots by exchanging selected information with peers. Inter-robot loop closures are discovered through communication (e.g. exchange of descriptors or keyframes), and the corresponding constraints are incorporated into each robot's local optimisation. Some systems employ consensus or distributed optimisation algorithms to ensure that the local estimates converge to a consistent global solution.

Hybrid and Hierarchical Architectures

Hybrid or hierarchical architectures combine elements of centralised and decentralised designs. For example, robots may form clusters, each with a cluster head that aggregates local information and communicates with other cluster heads

or a higher-level server. Alternatively, certain tasks (such as place recognition or map compression) may be centralised, while local trajectory optimisation remains on-board.

3

State of the Art

This chapter reviews representative C-SLAM systems with emphasis being placed on *backend optimisation and communication semantics* because these are the dimensions that most clearly differentiate centralised and decentralised designs while remaining comparable across sensor modalities. Broad surveys such as [1, 18, 4] cover the wider design space. The chapter is intentionally structured around the evaluation dimensions that later become first-class response variables: global consistency and accuracy, correction latency, bandwidth and communication structure, scalability with team size, and robustness under degraded networking.

1 Centralised C-SLAM

1.1 Architectural Patterns

Early work such as *C²TAM* [22] demonstrated the motivation for centralisation where computationally expensive mapping and optimisation can be offloaded to infrastructure, enabling lightweight agents to perform real-time tracking while still benefiting from global refinement. Related server-backed frameworks, such as the multi-robot pose graph server in [10], similarly highlight that central infrastructure can merge local maps even when robots do not share an initial global frame, provided inter-robot correspondences (e.g. visual loop closures) are available.

A recurring pattern in modern centralised systems is the separation of *local autonomy* from *global consistency*. In *CCM-SLAM* [24] and *CVI-SLAM* [14], each robot runs its local estimation loop continuously, while the server performs map merging and optimisation and returns drift-corrected poses. This design is attractive in practice because it degrades gracefully when communication is intermittent, local tracking continues, and global corrections are incorporated when connectivity becomes available again.

COVINS [25] advanced this approach; a central “map manager” receives keyframes from multiple agents and uses a unified place recognition pipeline (based on bag-of-words (*BoW*)) to detect both loop closures and inter-agent map overlaps. The server performs continuous pose graph optimisation and schedules full bundle adjustment periodically, balancing global consistency with real-time responsiveness. *COVINS-G* [21] generalises the same backend concept to accept heterogeneous frontends, which is a practical requirement in multi-robot teams where sensing stacks and odometry quality may differ across agents.

A second recurring pattern is *communication-aware summarisation*. Centralised systems rarely stream raw sensor data; instead, they transmit sparse keyframes, compact descriptors, downsampled point clouds, or selectively off-loaded subsets of observations, and the server returns comparatively small correction messages (e.g. optimised poses). Multi-modal systems show that the same principle extends beyond vision. *LAMP 2.0* [2] targets GPS-denied subterranean environments and integrates heterogeneous robots with a robust centralised backend. *CoLRIO* [32] combines LiDAR-inertial odometry with UWB ranging and sends downsampled point clouds, IMU pre-integrations, and range measurements to a central factor-graph optimiser. *AdaptSLAM* [5] makes the communication policy explicit by using an uncertainty-based rule to decide which keyframes to transmit, effectively treating bandwidth as a controllable resource.

1.2 Performance Implications

Centralised optimisation over all agents’ constraints tends to yield strong global consistency because the server can jointly exploit multi-view observations and inter-robot loop closures to correct drift that is unobservable from any single trajectory. This effect is visible in vision-centric systems such as *COVINS* [25] and in multi-modal fusion systems such as *CoLRIO* [32], where additional cross-robot constraints (e.g. UWB ranging) directly improve localisation in degraded visual conditions. Central servers can also apply computationally intensive refinement steps (e.g. periodic bundle adjustment in visual(-inertial) pipelines), which may be infeasible on resource-constrained robots.

Reported quantitative results illustrate these benefits, but remain difficult to compare directly across papers due to differing datasets, sensing stacks, and evaluation protocols. For instance, *COVINS* reports centimetre-level trajectory errors and low scale drift on EuRoC benchmarks [25], while *CoLRIO* reports that adding UWB ranging improves accuracy in degraded conditions (e.g. reducing ATE from 0.32 m to 0.23 m over 100 m trajectories) [32]. Central aggregation is

also particularly attractive for dense or shared reconstructions, where exchanging raw depth or point-cloud content peer-to-peer is impractical; systems such as *CCMD-SLAM* exemplify this direction [36].

At the same time, the literature converges on two limitations of the client-server paradigm. First, scalability is constrained by server computation and aggregate uplink bandwidth. As team size and map size grow, the server must ingest more keyframes, run more place recognition, and optimise a larger pose graph. Second, correction latency becomes a system-level variable, loop closures and global corrections are applied after a communication-and-optimisation cycle, so robots operate on local odometry between correction updates. This is not necessarily problematic, but it creates a trade-off between “how often to optimise and broadcast” and “how much bandwidth and compute to spend”, which is precisely the kind of trade-off that motivates controlled evaluation.

Reported communication characteristics in centralised systems also illustrate the typical asymmetry. For instance, *COVINS* reports keyframe transmission at 5 Hz and pose-correction return messages at 2 Hz, with an uplink bandwidth on the order of hundreds of kB/s per agent (reported around ~ 493 kB/s) and a much smaller downlink for corrections (reported around ~ 2.3 kB/s) [25]. LiDAR-centric systems such as *CoLRIO* report higher per-keyframe payloads due to point-cloud components (e.g. ~ 242 kB keyframes and ~ 600 kB/s per robot), yet still rely on downsampling and selective transmission to keep traffic manageable [32]. These figures are not directly comparable across papers because datasets, sensing stacks, and implementation details differ; nevertheless, they consistently show that centralised designs depend on sustained uplink capacity and that server-side processing time and network conditions jointly shape correction delays.

2 Decentralised/Distributed C-SLAM

2.1 Architectural Patterns

The foundational distributed factor-graph architecture was established by *DDF-SLAM* [8, 7], which formalised how local estimation problems can be coupled through selective factor exchange while preserving autonomy. Contemporary systems such as *DCL-SLAM* [33] and *Swarm-SLAM* [16] build on this principle by making collaboration opportunistic where agents exchange information with neighbours when connectivity is available, without requiring persistent global communication.

Because decentralised systems cannot rely on a high-capacity uplink to a server, they tend to treat bandwidth as a primary design constraint. Visual systems such as *Data-Efficient Decentralised V-SLAM* [6] pioneered a two-stage protocol in which robots first exchange compact descriptors to propose candidate loop closures and transmit heavier keyframe data only when a promising correspondence is detected. *DOOR-SLAM* [17] follows the same idea and focuses on sharing only the information required for inter-robot loop closures, combined with explicit consistency filtering. *Kimera-Multi* [3, 30] similarly leverages compact

bag-of-words descriptors and sparse representations during robot rendezvous, enabling robust place recognition without continuous large data exchange.

LiDAR-centric decentralised systems adopt analogous strategies. *DCL-SLAM* [33] employs lightweight LiDAR descriptors such as LiDAR-Iris and shares keyframe scans and loop constraints on demand rather than continuously. *Swarm-LIO2* [34] is designed for UAV swarms and broadcasts minimal state packets (identity, current pose, mutual observations), enabling plug-and-play collaboration under strict bandwidth budgets. *D²SLAM* [31] introduces an adaptive policy that distinguishes near-field and far-field collaboration. Nearby agents exchange richer information for high-accuracy fusion, while distant agents receive compressed summaries to preserve scalability.

A second cornerstone is *distributed optimisation and outlier rejection*. Without central arbitration, incorrect inter-robot loop closures can corrupt the estimate unless filtered robustly. *Kimera-Multi* couples collaboration with robust block coordinate descent (RBCD) for distributed pose graph optimisation and employs a max-clique-based strategy to filter inconsistent loop closures. Several systems incorporate explicit geometric consistency checks such as PCMS (e.g. *DOOR-SLAM* [17] and *DCL-SLAM* [33]) to validate inter-robot loop closures locally before accepting them into the distributed optimisation. These verification steps are not optional details: they are core enablers of decentralisation because communication is asynchronous and incorrect constraints cannot be “fixed” by a single trusted server.

2.2 Performance Implications

Decentralised systems avoid a single network bottleneck and can achieve low frontend latency because odometry and local estimation remain onboard. The trade-off is that global consistency is reached through iterative exchange and distributed optimisation rather than a single joint solver step. For example, *D²SLAM* reports visual-inertial odometry and loop closure processing below 20 ms per frame on standard hardware [31], while *Kimera-Multi* reports that its distributed optimiser can provide rapid approximate solutions and higher-quality solutions with longer convergence times [3]. Such results illustrate a general property of decentralised backends that the time-to-global-consistency depends on both the optimisation schedule and the communication opportunities between agents.

In return, decentralised architectures can drastically reduce communication requirements through selective exchange. *Data-Efficient Decentralised V-SLAM* transmits compact descriptors first (reported around ~ 200 B) and sends full keyframes only when necessary (reported ~ 50 – 100 kB), yielding large reductions relative to naive keyframe broadcasting [6]. *DOOR-SLAM* reports similarly compact inter-robot exchanges when a loop closure is confirmed and filtered (reported < 20 kB per successful exchange) [17]. Swarm-scale systems emphasise minimal packet designs. *Swarm-LIO2* targets per-agent traffic below 10 kB/s [34], and *D²SLAM* distinguishes near-field sharing of richer packets (reported ~ 100 kB keyframes) from far-field sharing of compressed summaries to preserve scala-

bility [31]. These reported behaviours consistently indicate that decentralised systems can bound bandwidth usage, at the cost of higher onboard computation and more complex coordination and verification protocols.

3 Comparative Analysis

The state of the art portrays centralised and decentralised C-SLAM as two ends of an architectural spectrum, with hybrid methods interpolating between them [16, 31, 5]. From the perspective of this research thesis, the key point is that performance differences are not purely algorithmic; they are emergent properties of how optimisation, communication, and system scheduling interact. Table 3.1 lists key characteristics that enable comparison between the architectures.

Dimension	Centralised SLAM	Decentralised SLAM
Global Accuracy	High	Moderate to High
Latency	Higher (offloaded to server)	Lower (local)
Scalability	Limited by server capacity and bandwidth bottleneck	Scales with team size; depends on network topology
Communication	High uplink; asymmetric patterns	Low; bounded; peer-to-peer
Robustness	Central server as single point of failure	High resilience; graceful degradation with robot failures

Table 3.1: Comparison of centralised and decentralised C-SLAM architectures. These trends are consistent across visual, LiDAR, and multi-modal systems.

Despite extensive research, three limitations recur across published C-SLAM systems.

(1) Limited controlled comparison across architectures. Most papers demonstrate a single pipeline end-to-end and compare against baselines that differ in more than one aspect (frontend, sensing, map representation, compute platform, or parameterisation). As a result, it is difficult to attribute observed performance differences to the architectural variable itself rather than to confounding factors.

(2) Networking is usually a background assumption rather than a first-class experimental factor. While many systems motivate their design via bandwidth reduction or resilience to intermittent links, evaluations often assume good connectivity or report results without systematic control of packet loss, added latency, bandwidth caps, or temporary partitions. Consequently, it remains unclear how robust different backends are under realistic communication stressors and how correction latency depends on network conditions.

(3) Metrics and reporting are fragmented. Accuracy is typically reported (e.g. ATE), but latency, bandwidth usage, CPU/memory footprints,

and time-to-global-consistency are not consistently measured or defined, and reporting is often system-specific. This makes it difficult to compare approaches or to reason about architectural trade-offs.

These observations motivate a controlled evaluation in which architectural choice is treated as an experimental factor and networking behaviour is explicitly modelled.

Methodology

1 Experimental Design

This section outlines an experimental research design to evaluate the performance trade-offs between C-SLAM backend architectures. A central requirement is that different architectures are assessed under *identical experimental conditions* to ensure a fair, reproducible comparison. The study adopts a controlled, simulation-based methodology in which all pipeline components that are independent of the architectural choice are treated as fixed. Only the *backend optimisation architecture* and its associated *communication semantics* are varied.

In a complete C-SLAM system, each agent would run a local frontend (sensor ingestion, feature extraction/scan matching, local loop closure, and keyframe selection). In this work, frontend processing is abstracted by dataset playback so that backend effects can be isolated without conflating them with frontend design choices.

1. Dataset playback and constraint-stream replay
2. Backend ingestion and optimisation
3. Pose update dissemination
4. Metric logging and post-processing evaluation.

To isolate the effects of architectural selection and structure the evaluation, we define a design space over four groups of experimental variables that capture key dimensions of computation, communication, and environmental variability while remaining abstracted from implementation-specific details:

- *Architecture*: \mathcal{A}
- *Team Size*: \mathcal{N}
- *Communication Profile (QoS)*: \mathcal{Q}
- *Network Impairments*: \mathcal{I}

This yields the core experimental design space as the Cartesian product

$$\mathcal{D} = \mathcal{A} \times \mathcal{N} \times \mathcal{Q} \times \mathcal{I}. \quad (4.1)$$

A full combinatorial evaluation over \mathcal{D} would comprise $|\mathcal{A}| \cdot |\mathcal{N}| \cdot |\mathcal{Q}| \cdot |\mathcal{I}|$ conditions, which is computationally expensive. To control the combinatorial growth while maintaining interpretability, a two-stage strategy is adopted:

1. **Baseline Grid**: \mathcal{Q} and \mathcal{I} are fixed to reference settings and compare architectures under clean network conditions. This establishes baseline performance characteristics.
2. **Targeted Sweeps**: By varying one variable group at a time (e.g., \mathcal{Q} , \mathcal{I} , \mathcal{N}) while holding others at baseline. This characterises:
 - *QoS sensitivity*: how delivery guarantees and history settings affect latency, reliability, and stability
 - *Network robustness*: response to packet loss, delay, bandwidth constraints, and temporary partitions
 - *Scalability with team size*: effects of increasing the number of agents on coordination and computational load

2 State Representation and Estimation Variables

To compare backend architectures under a *fixed* estimation problem, all systems in this thesis solve the same pose graph maximum-likelihood objective and differ only in how optimisation and information exchange are realised. We adopt the factor-graph formulation and the generic NLLS template from Chapter 2 (Section 2.1.2 and Eq. (2.6)) and instantiate it here to the specific multi-robot, pose-only setting used in the experiments.

2.1 Pose Variables

Each robot maintains keyframe poses $x_i^{(r)} \in SE(d)$ with $d \in \{2, 3\}$ depending on the dataset. A pose is written as $x = (t, r)$ with translation $t \in \mathbb{R}^d$ and orientation $r \in SO(d)$; the manifold notation and parameterisations are introduced in Section 2.1. In the implementation, orientations may be stored in an equivalent representation (e.g., quaternions for $SO(3)$), but the estimation problem is defined on $SE(d)$.

2.2 Multi-Robot Global State

Consider a team of N robots. For robot $r \in \{1, \dots, N\}$, let T_r denote the index set of keyframes (pose nodes) used by the backend, and let $x_i^{(r)}$ be the pose of robot r at keyframe $i \in T_r$. The global pose graph state is the set of all robot poses

$$X = \left\{ x_i^{(r)} \mid r \in \{1, \dots, N\}, i \in T_r \right\}. \quad (4.2)$$

All architectural variants considered in this work operate on the same conceptual state X ; differences arise from how X is partitioned, updated, and synchronised.

2.3 Constraint Sets and Pose Graph

The frontend (or, in this experimental setup, dataset playback) provides relative-pose constraints between keyframes. We distinguish:

- **Odometry / motion constraints** between successive keyframes on each robot, collected in \mathcal{E}_{odo} ,
- **Intra-robot loop closures** within a robot’s trajectory, collected in $\mathcal{E}_{\text{intra}}$, and
- **Inter-robot loop closures** coupling poses across robots, collected in $\mathcal{E}_{\text{inter}}$.

The full edge set is their union $\mathcal{E} \triangleq \mathcal{E}_{\text{odo}} \cup \mathcal{E}_{\text{intra}} \cup \mathcal{E}_{\text{inter}}$, optionally augmented with a unary *anchoring* factor on a designated reference pose to remove gauge freedom.

Each constraint $(i, j) \in \mathcal{E}$ is represented by a relative-pose measurement z_{ij} with likelihood

$$f_{ij}(x_i, x_j) \propto \exp\left(-\frac{1}{2} \|z_{ij} - h(x_i, x_j)\|_{\Omega_{ij}}^2\right), \quad (4.3)$$

where $h(\cdot)$ predicts the relative transformation implied by the current pose estimates and Ω_{ij} is the information matrix. The Mahalanobis norm is $\|e\|_{\Omega_{ij}}^2 \triangleq e^\top \Omega_{ij} e$.

2.4 Reference Optimisation Problem

With independent, zero-mean Gaussian noise, maximising the joint likelihood over the pose variables yields the pose graph NLLS objective. We restate it here in full because it serves as the common reference objective across all architectures in Section 4.3:

$$\hat{\mathcal{X}} = \arg \max_{\mathcal{X}} p(\mathcal{Z} \mid \mathcal{X}) = \arg \min_{\mathcal{X}} \sum_{(i,j) \in \mathcal{E}} \|\mathbf{z}_{ij} - h(\mathbf{x}_i, \mathbf{x}_j)\|_{\boldsymbol{\Omega}_{ij}}^2, \quad (4.4)$$

where \mathcal{X} denotes the set of pose variables (i.e., X above) and \mathcal{Z} the set of all relative-pose measurements. Compared to the generic template in Eq. (2.6), the “where we change stuff” specialisation is explicit: (i) we estimate a pose-only state (no explicit landmark variables), (ii) we couple robots through $\mathcal{E}_{\text{inter}}$, and (iii) we fix the global reference frame through an anchoring factor.

Equation (4.4) is the estimation problem shared by all pipelines in this thesis; the remaining sections specify how centralised and decentralised backends realise this optimisation under different communication and computation constraints.

3 C-SLAM Backend Architectures

Within the experimental design space of Section 4.1, the architectural variable \mathcal{A} has two values: *centralised* and *decentralised*. Both variants aim to solve the same pose graph objective in Equation (4.4) (a special case of the general template in Chapter 2, Equation 2.7); the comparison therefore isolates *where* optimisation is executed and *what* information is communicated to achieve a consistent estimate. This section specifies how each variant instantiates the backend, including where optimisation is executed and how pose updates and auxiliary information are communicated.

3.1 Centralised C-SLAM Backend

In the centralised architecture, a dedicated backend server maintains a single global pose graph over the multi-robot state X . Each robot (or a dataset replay process) periodically sends *constraint batches* to the server. These batches contain odometry constraints, intra-robot loop closure constraints, and inter-robot loop closure constraints. The server aggregates the received constraints into one global graph and runs joint pose graph optimisation over the full state X . Where enabled, updated pose estimates are broadcast back to each agent as global state updates. In this work, inter-robot loop closures are treated as pre-computed constraints provided by the dataset; data association and loop-closure verification are therefore outside the scope of the backend comparison.

Let $\mathcal{C} \subseteq \mathcal{E}$ denote the set of all intra- and inter-robot constraints that have

been received by the server. The centralised backend solves

$$X^* = \arg \min_X \sum_{(i,j) \in \mathcal{C}} e_{ij}(X)^\top \Omega_{ij} e_{ij}(X), \quad (4.5)$$

Equation (4.5) is a special case of Eq. (4.4), where the full edge set \mathcal{E} is replaced by the subset \mathcal{C} available at the server at that time.

where $e_{ij}(X) = z_{ij} - h(x_i, x_j)$ is the residual associated with constraint (i, j) as in Equation (4.3).

3.2 Decentralised Distributed SLAM Backend

In the decentralised architecture, no global server exists. Each robot maintains its own local pose graph and participates in a distributed optimisation process via peer-to-peer communication. The objective in Equation (4.4) is the same as in the centralised case, but it is approximated through local computations and information exchange between neighbouring robots.

For robot $r \in \{1, \dots, N\}$, let

$$G_r = (V_r, E_r) \quad (4.6)$$

denote its local pose graph, where V_r contains the local pose variables $x_i^{(r)}$ for $i \in T_r$ and E_r contains odometry and intra-robot loop closure constraints generated by robot r . Inter-robot loop closures introduce cross-graph coupling constraints between robots r and s of the form

$$f_{rs}(x_i^{(r)}, x_j^{(s)}), \quad (4.7)$$

which connect poses belonging to different local graphs. Using this notation, the global optimisation problem can be written as

$$\min_{\{x^{(r)}\}} \left[\sum_{r=1}^N f_r(x^{(r)}) + \sum_{(r,s)} f_{rs}(x^{(r)}, x^{(s)}) \right], \quad (4.8)$$

Equation (4.8) is not a different estimation objective; it is Eq. (4.4) rewritten by grouping terms into per-robot local objectives $f_r(\cdot)$ and cross-robot coupling terms $f_{rs}(\cdot, \cdot)$. Thus, the methodological change from Section 4.3.1 to Section 4.3.2 lies in the computation and communication pattern, while the underlying maximum-likelihood objective remains the same.

where $f_r(\cdot)$ collects all local constraints in E_r and $f_{rs}(\cdot)$ collects all cross-robot coupling constraints between robots r and s .

4 Communication Model and Network Variables

The backend architectures in Section 4.3 are realised on top of a message-oriented communication layer. This section formalises the communication patterns induced by each architecture and the way network conditions are modelled in the experiments. In the design space \mathcal{D} (Section 4.1), these aspects are captured by the *Communication Profile (QoS)* variable \mathcal{Q} and the *Network Impairments* variable \mathcal{I} .

4.1 Communication Model

Two distinct communication models are employed, corresponding to the centralised and decentralised backend architectures.

In the centralised configuration, communication follows an asymmetric client-server pattern where all robots transmit constraint batches to the server (**Uplink**). These batches include odometry constraints and loop closure constraints provided by the dataset (or produced by a frontend in a full system). Where enabled, the server transmits global pose corrections (or updated keyframe poses) back to each robot (**Downlink**).

In the decentralised configuration, no central server exists and communication is peer-to-peer. Robots exchange interface messages that summarise boundary state estimates (e.g. pose mean and covariance for poses involved in inter-robot loop closures), enabling distributed optimisation without broadcasting a full global state. The communication model is therefore local and topology-dependent.

Messages exchanged in both architectures are timestamped at send and at ingestion time, enabling measurement of end-to-end delays from constraint publication to backend use, as well as message inter-arrival times and effective throughput (bytes per second) per link and per robot. These measurements are used in the evaluation to quantify communication latency and bandwidth usage under different combinations of \mathcal{A} , \mathcal{Q} , and \mathcal{I} .

4.2 Network Impairment Model

To evaluate the robustness of C-SLAM architectures under adverse communication conditions, controlled network impairments are introduced in the experimental setup. The Network Impairments variable \mathcal{I} parameterises the following impairment types:

- **Stochastic loss:** Messages are randomly dropped according to a specified loss rate, modelling unreliable links and congestion.
- **Burst loss:** Messages are dropped in periodic bursts, modelling transient interference and contention.
- **Blackouts / partitions:** Scheduled sender or receiver blackouts emulate temporary disconnections (including asymmetric communication failures).

- **Bandwidth caps:** Per-sender throughput limits are enforced via token-bucket shaping, which introduces send delays consistent with a constrained channel.

Impairments are applied in a controlled and repeatable manner by a network emulation layer at message send time. For a given impairment configuration \mathcal{I} , the same impairment parameters are applied across both architectures to ensure architectural fairness.

In practice, the impairment mechanism supports different *application modes* depending on which communication edges are being studied: impairments can be applied to (i) the factor-publisher uplink (constraint availability), (ii) backend-internal communication (decentralised peer-to-peer interface exchange and/or centralised map downlink), or (iii) both. Fair comparisons therefore condition on a fixed impairment mode and parameter set, and results are interpreted accordingly.

By combining the communication models of Section 4.4.1 with the impairment levels defined here, the experimental framework systematically explores how architectural choices (\mathcal{A}) interact with QoS settings (\mathcal{Q}) and network impairments (\mathcal{I}) in terms of latency, communication load, scalability, and estimation accuracy.

5 Performance metrics

For each experimental configuration in the design space \mathcal{D} , a set of response variables is evaluated to capture system performance in terms of latency, estimation accuracy, and resource usage. These metrics provide the quantitative basis for comparing centralised and decentralised architectures under varying team sizes and network conditions. The metrics are grouped into three categories: *Timing metrics*, *Accuracy metrics* and *Resource metrics*.

5.1 Timing metrics

Timing metrics quantify (i) how quickly a backend incorporates newly ingested constraints and (ii) how quickly corrections become visible to the team. To make compute time and dissemination time comparable across architectures, we distinguish backend solve latency from (optional) update broadcast latency.

Ingest-to-optimisation latency L_{opt} . For the centralised backend, where each ingest event corresponds to a clearly defined server-side batch, we additionally report the elapsed time between ingestion of a constraint batch and completion of the corresponding optimisation update:

$$L_{\text{opt}}(b) = t_{\text{optimization_end}}(b) - t_{\text{ingest}}(b), \quad (4.9)$$

where $t_{\text{ingest}}(b)$ is the timestamp at which the backend accepts batch b and $t_{\text{optimization_end}}(b)$ is the timestamp at which the optimisation update that incorporates b completes. For decentralised runs there is no single, architecture-

agnostic notion of “backend ingest of batch b ” because each agent ingests both local factors and asynchronous peer updates; in this thesis we therefore treat ingest-to-optimisation as architecture-specific and rely on D_{opt} as the cross-architecture compute indicator.

Ingest-to-broadcast latency L_{global} (centralised) For the centralised architecture, the server may additionally publish a global update (map downlink) to make the updated state available to robots. The corresponding dissemination latency is defined as

$$L_{\text{global}}(b) = t_{\text{broadcast}}(b) - t_{\text{ingest}}(b), \quad (4.10)$$

where $t_{\text{broadcast}}(b)$ denotes the time at which the updated state is published. In decentralised runs there is no single broadcast event; therefore L_{global} is not defined and settling behaviour is characterised via stabilisation-based metrics (T_{LC} and T_{conv}).

Stabilisation criterion Several timing KPIs require a backend-agnostic notion of “the estimate has settled”. Let $\mathbf{t}_i^{(u)} \in \mathbb{R}^d$ denote the translation component of pose x_i at backend update/round index u . We define the maximum translation change between successive updates as

$$\Delta t_{\text{max}}^{(u)} = \max_i \left\| \mathbf{t}_i^{(u)} - \mathbf{t}_i^{(u-1)} \right\|_2. \quad (4.11)$$

Unless stated otherwise, stabilisation is declared when $\Delta t_{\text{max}}^{(u)} \leq \epsilon$ for S consecutive updates, with $\epsilon = 10^{-3}$ and $S = 3$.

Loop-closure correction time T_{LC} This metric measures how long it takes for an inter-robot loop closure to be reflected in a stable estimate. For a loop-closure event c :

$$T_{\text{LC}}(c) = t_{\text{stable}}(c) - t_{\text{det}}(c), \quad (4.12)$$

where $t_{\text{det}}(c)$ is the time at which the loop-closure constraint is ingested and $t_{\text{stable}}(c)$ is the timestamp of the first update that begins a stable window according to the stabilisation criterion above. This captures both optimisation time and (where applicable) communication delays for the correction to propagate.

Time-to-convergence T_{conv} For long runs, we measure the time required for the system to reach a globally stable pose graph under the same stabilisation criterion. Let u^* be the first update index at which $\Delta t_{\text{max}}^{(u)} \leq \epsilon_{\text{conv}}$ holds for S consecutive updates. The time-to-convergence is then

$$T_{\text{conv}} = t(u^*) - t_{\text{start}}, \quad (4.13)$$

where $t(u^*)$ is the timestamp of the first stable update and t_{start} is the start time of the run. Unless stated otherwise, we use $\epsilon_{\text{conv}} = 10^{-3}$ and $S = 3$ to match

the KPI derivation implemented in Chapter 5.

5.2 Accuracy Metrics

Accuracy is evaluated in terms of both global trajectory consistency and local drift. Absolute Trajectory Error (ATE) and Relative Pose Error (RPE) follow the definitions in [28], for each agent, and aggregate these across the team.

Absolute Trajectory Error (ATE) ATE measures the deviation between estimated and ground-truth trajectories after alignment. Because COSMO-Bench provides metric LiDAR trajectories, scale is fixed to $s^* = 1$ and alignment is performed using a rigid transform in $SE(3)$ (rotation \mathbf{R}^* and translation \mathbf{t}^*) computed via Umeyama’s method. The aligned RMSE ATE for agent a is

$$\text{ATE}_{\text{RMSE}}^{(a)} = \sqrt{\frac{1}{T_a + 1} \sum_{k=0}^{T_a} \left\| \mathbf{x}_k^{(a), \text{gt}} - \left(\mathbf{R}^* \mathbf{x}_k^{(a)} + \mathbf{t}^* \right) \right\|_2^2}. \quad (4.14)$$

Relative Pose Error (RPE) RPE quantifies local drift between pairs of poses separated by a fixed time interval Δ . For agent a , let $\mathbf{t}_{k:\Delta}^{(a)}$ and $\mathbf{t}_{k:\Delta}^{(a), \text{gt}}$ denote the estimated and ground-truth relative translations between times k and $k + \Delta$. The translational component of the relative error is

$$r_k^{(a)}(\Delta) = \left\| \mathbf{t}_{k:\Delta}^{(a), \text{gt}} - \mathbf{t}_{k:\Delta}^{(a)} \right\|_2, \quad (4.15)$$

and the RMSE RPE for agent a is

$$\text{RPE}_{\text{RMSE}}^{(a)}(\Delta) = \sqrt{\frac{1}{T_a - \Delta} \sum_{k=0}^{T_a - \Delta} \left(r_k^{(a)}(\Delta) \right)^2}. \quad (4.16)$$

RPE is evaluated for multiple values of Δ to capture both short- and long-horizon drift characteristics.

5.3 Resource Metrics

Resource metrics assess the computational and communication costs incurred during C-SLAM. They are particularly relevant for comparing architectures in terms of scalability and efficiency.

CPU Utilisation ρ_{CPU} CPU utilisation is sampled periodically at the process level for the processes involved in each run (e.g. constraint publisher and backend processes). Summary statistics are computed over the evaluation window $[0, T_{\text{eval}}]$, and multi-process backends are reported either per-process or as an aggregate across processes, depending on the available logging.

Peak Memory Usage M_{peak} Peak memory usage is defined as the maximum resident set size (RSS) observed during execution for the processes involved in the run. For multi-process backends, the peak can be reported per-process and/or as an aggregate across processes, depending on the available logging.

Bandwidth Usage BW Bandwidth usage quantifies the total amount of data transmitted per agent over the course of a run. For each agent a , we compute

$$BW^{(a)} = \frac{1}{T_{\text{eval}}} \int_0^{T_{\text{eval}}} \sum_c B_c^{(a)}(t) dt, \quad (4.17)$$

where $B_c^{(a)}(t)$ is the instantaneous byte rate on communication channel c at time t , and T_{eval} is the evaluation horizon. In practice, this integral is approximated from logged message sizes and timestamps. We additionally report an effective delivery rate, defined as the ratio of successfully received to sent messages, to account for packet loss under impaired network conditions.

Together, these timing, accuracy, and resource metrics provide a multi-dimensional view of system performance, enabling a structured analysis of the trade-offs between centralised and decentralised C-SLAM architectures under varying team sizes, QoS settings, and network impairment levels.

Implementation

This chapter explains how the experimental methodology is realised in software. The focus is on (i) enforcing a *comparability contract* between architectures by replaying an identical factor stream, (ii) implementing a centralised and a decentralised backend on top of a shared factor representation, (iii) realising controlled communication conditions via ROS 2 QoS and message-level impairments, and (iv) instrumenting the system to produce reproducible KPI logs for the evaluation chapter.

1 Dataset and Frontend

A fair architectural comparison requires that both backends consume an identical frontend output. Although the S3E dataset [11] was initially considered due to its rich multimodal sensing, its computational and storage demands made it impractical for the available experimental setup. More importantly, implementing or modifying a frontend would have introduced additional degrees of freedom that are orthogonal to the architectural research questions.

The **COSMO-Bench** dataset [19] was selected because it provides a standardised C-SLAM frontend and a benchmark suite of 24 sequences with annotated intra-/inter-robot loop closures and communication scenarios (WiFi and ProRadio). Each sequence is distributed in the JRL (JSON Robot Log) format, which directly encodes the factor stream consumed by both architectures.

Frontend Abstraction The “common SLAM frontend” referenced in the methodology (Section 4.1) is instantiated by COSMO-Bench’s preprocessing pipeline. Odometry constraints, intra-robot loop closures, and inter-robot loop closures are already detected and validated. The resulting JRL files provide:

- initial pose estimates for all robot keyframes,
- prior factors anchoring each robot’s trajectory,
- between-factors for odometry and loop closures (intra- and inter-robot),
- ground-truth trajectories for evaluation,
- optional outlier annotations (not used in the main evaluation unless stated otherwise).

Comparability Contract Both backends ingest the same JRL factors through the shared loader `c_slam_common/loader.py`. As a result, any measured differences in latency, bandwidth, or estimation behaviour are attributable to the backend architecture and the configured network conditions, not to differences in frontend processing.

2 System Overview

The framework consists of four subsystems: (i) a shared common library for dataset parsing, factor construction, and KPI logging, (ii) a centralised backend implementing global incremental optimisation, (iii) a decentralised backend implementing DDF-SAM-style distributed coordination, and (iv) a ROS 2 middleware layer providing transport, QoS control, and message-level impairments.

The reference implementation targets Ubuntu 22.04 LTS with ROS 2 Humble and Python 3.10. GTSAM ($\geq 4.2.0$) is used via its Python bindings for factor-graph construction and iSAM2 optimisation. All experiments execute on a single physical machine, while ROS 2 multi-process execution emulates a distributed deployment and allows controlled communication conditions. Executing both architectures on one host introduces an architecture-induced confound: OS scheduling contention (especially in the decentralised multi-process case) can influence wall-clock solver timing; to contextualise timing KPIs, CPU and memory utilisation are sampled and exported alongside evaluation logs (Section 5.7.4).

3 Operationalisation of the Experimental Design Space

The experimental design space \mathcal{D} introduced in the methodology (Section 4.1) is instantiated through runtime configuration knobs controlling: backend architecture, team size, ROS 2 QoS, and network impairments. The implementation

Path	Purpose
<code>main.py</code>	CLI entry point: selects backend, dataset, configuration, and output paths
<code>c_slam_common/</code>	Shared utilities: JRL parsing, factor graph building, noise configuration, KPI/event logging, bandwidth/latency tracking, resource monitoring, helpers
<code>c_slam_central/</code>	Centralised backend: global iSAM2 optimiser, ROS 2 factor ingestion, optional downlink/ACKs, KPI hooks
<code>c_slam_decentral/</code>	Decentralised backend: per-agent iSAM2, interface summarisation/exchange, multi-process runner, KPI exports
<code>c_slam_ros2/</code>	ROS 2 middleware: factor/interface/map messages, QoS configuration, impairment policy, simulation time helpers
<code>tools/</code>	Orchestration scripts and KPI derivation/export utilities
<code>dataset/</code>	Example JRL sequences used for development (COSMO-Bench factor streams)

Table 5.1: Repository layout of the experimental framework.

exposes these knobs via CLI flags and environment variables to enable reproducible sweeps.

End-to-end Run Workflow A run replays a JRL factor stream over ROS 2 (Subsection 5.3.1) so that both backends observe the same sequence of priors and between-factors. The selected backend then (i) performs incremental global updates (centralised) or (ii) executes a fixed number of DDF rounds (decentralised), exporting estimates and a timestamped KPI event log. Derived KPIs are computed offline from the event log by `tools/kpi_derive.py`.

Reproducibility Artefacts Each run records resolved configuration in `run_manifest.json`. Orchestrated runs additionally write the resolved orchestrator config and command list (e.g., `orchestrate_config.json`, `orchestrate_commands.json`) into the export directory, allowing exact reconstruction of command lines and environment variables.

3.1 ROS 2 Factor Streaming

A ROS 2 middleware layer emulates real-time factor streaming from each agent without requiring live robots. The factor publisher `tools/ros2_factor_publisher.py` publishes per-robot batches on topics `<prefix>/<robot_id>`.

Design variable	Implementation knobs
Architecture \mathcal{A}	<code>main.py --backend</code> with values centralised or decentralised . Orchestrated runs set the same via <code>tools/orchestrate.py / new_eval</code> configs.
Team size N	Derived from the selected JRL sequence: robot IDs are extracted from initialisation blocks (<code>build_key_robot_map</code>). The decentralised runner spawns one process per robot (<code>c_slam_decentral/mp_runner.py</code>). COSMO-Bench sequences <code>r3_*</code> , <code>r4_*</code> , <code>r5_*</code> realise $N \in \{3, 4, 5\}$.
QoS profile \mathcal{Q}	<code>--qos-reliability</code> , <code>--qos-durability</code> , <code>--qos-depth</code> , applied consistently to factor batches, interface messages, and optional downlink (Section 5.6).
Impairments \mathcal{I}	JSON profile via <code>C_SLAM_IMPAIR</code> or <code>C_SLAM_IMPAIR_FILE</code> ; instantiated by <code>ImpairmentPolicy</code> (<code>c_slam_ros2/impair.py</code>) and injected identically across processes.

Table 5.2: Operationalisation of the design space into concrete runtime knobs.

Batches are JSON-encoded (`c_slam_ros2/factor_batch.py`) and transmitted as `std_msgs/UInt8MultiArray`. Each batch includes lightweight metadata required for instrumentation: `message_id` (UUID), `send_ts_mono` (monotonic timestamp), and `send_ts_wall` (wall-clock timestamp). Robot identity is implied by the topic name and not duplicated in the payload.

Factor payload The stream supports `PriorFactorPose3` and `BetweenFactorPose3` objects (rotation, translation, covariance, keys, stamp). These are translated into GTSAM factors by the backend-specific graph builders. Completion is signalled via `<prefix>/control` messages `DONE:<robot_id>` and a final `DONE_ALL` marker.

4 Centralised Backend

The centralised backend aggregates factors from N robots into a single global factor graph and incrementally optimises the joint state using iSAM2. The reference implementation is `c_slam_central/central_backend.py`. Figure 5.1 summarises the interaction between ROS 2 streaming and the central server.

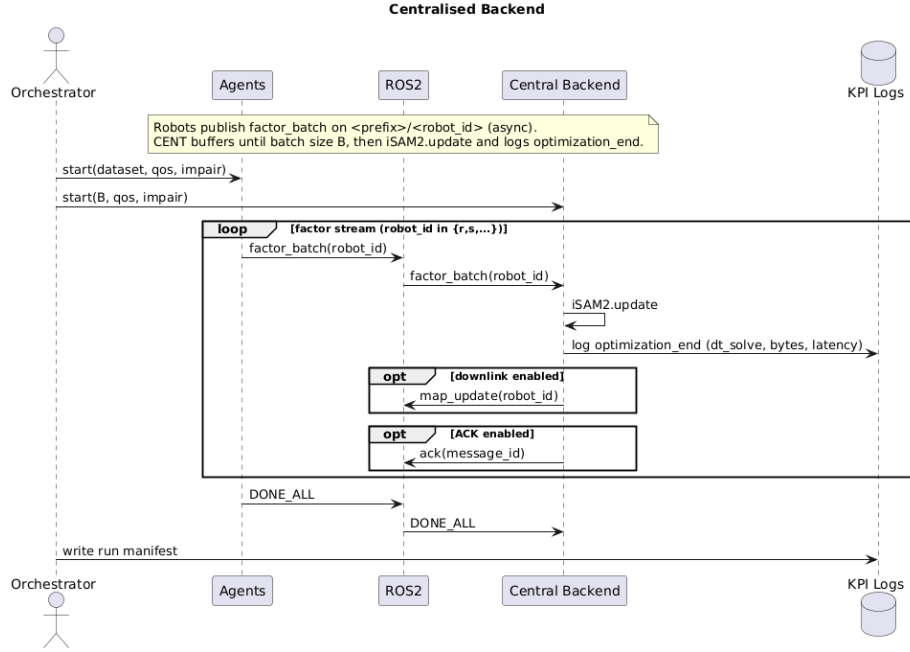


Figure 5.1: Sequence diagram depicting agent-central server interaction.

4.1 Factor Ingestion and Batching

The `ROS2FactorSource` (`c\slam\ros2\factor_source.py`) subscribes to `<prefix>/<robot_id>` for all robots. Incoming batches are decoded and annotated with message-level metadata used downstream for KPI logging:

- `__ros2_topic` (source topic),
- `__ros2_msg_bytes` (serialised payload size),
- `__ros2_msg_id` (publisher UUID),
- `send_ts_mono`, `send_ts_wall` (publisher timestamps).

The publisher can bundle multiple factors per ROS 2 message (`--batch-size`). On the server, decoded factors are accumulated until a solver batch threshold is reached (`main.py --central-batch-size`). Each solver batch produces a per-batch graph and initial-value deltas, which are then applied in one `iSAM2` update. Ingestion terminates when `DONE_ALL` is received or when an idle timeout expires (`--central-idle-timeout`).

Initial Values Initial estimates are obtained from JRL initialisation blocks and stored in `init_lookup`. If a key is missing, the implementation inserts

an identity pose as a fallback and logs a warning, ensuring the graph remains well-formed without factor deferral logic.

4.2 Solver: Incremental Optimisation with iSAM2

iSAM2 is configured through `ISAM2Params` in `c_slam.common/isam2.py`. The main exposed knobs are the relinearisation threshold (`--relin-th`) and skip interval (`--relin-skip`). Algorithm 1 summarises the incremental update loop used in the experiments.

Algorithm 1 Centralised iSAM2 backend: incremental optimisation loop

Require: Stream of factors \mathcal{F} , initial pose lookup `init_lookup`, batch size B

Ensure: Global estimate \hat{X}

- 1: Initialise GRAPHBUILDER and iSAM2
 - 2: Accumulate decoded factors into a solver batch of size B
 - 3: For each completed batch: build (G_b, V_b) and call `iSAM2.UPDATE(G_b, V_b)`
 - 4: On numerical/duplicate-key failure: rebuild iSAM2 from accumulated global graph and retry once
 - 5: After each successful update: extract \hat{X} and emit KPI events for optimisation timing and batch attribution
 - 6: **return** \hat{X}
-

4.3 Optional Downlink and ACK Instrumentation

In a deployed centralised system, the server would broadcast the updated global estimate to robots. The framework supports an optional “map downlink” publication (`--emit-map-downlink`) implemented in `c_slam_ros2/map_broadcast.py`. When enabled, downlink traffic is subjected to the same QoS and impairment settings as other topics, and is measured from actual serialised ROS 2 payload sizes.

For delivery-rate diagnostics, optional factor-batch acknowledgements can be enabled (`--emit-factor-acks`). ACK messages bind processed `message_ids` to the solver timeline and allow distinguishing “published” from “observed and processed” traffic under impairments.

Payload Accounting For interpretability, the framework may compute a rough lower-bound estimate of the downlink payload assuming binary float64 encoding of pose means (translation + quaternion). However, all reported bandwidth statistics in the evaluation are based on measured serialised message sizes (`_ros2_msg_bytes` and recorded publish sizes), which capture JSON overhead and key strings.

5 Decentralised Backend

The decentralised backend implements a DDF-SAM-style distributed optimiser [8, 7]. Each of the N agents maintains a local factor graph and exchanges separator summaries with peers. There is no central coordinator; global consistency emerges through repeated local optimisation and information exchange. Figure 5.2 shows the high-level interaction.

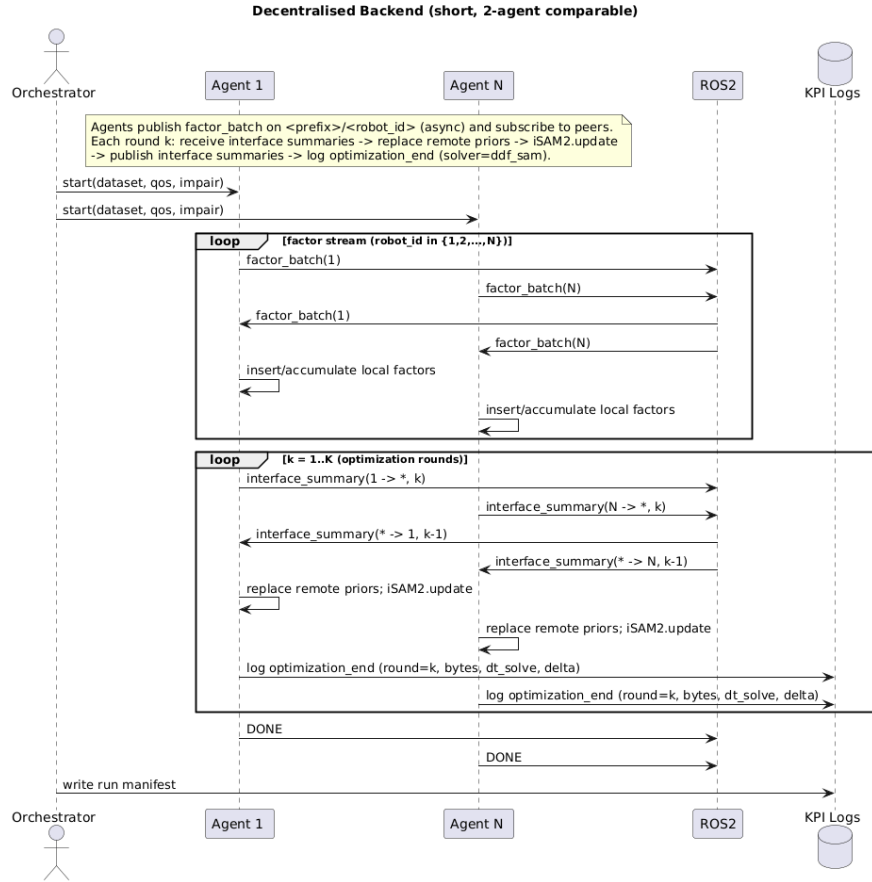


Figure 5.2: Sequence diagram depicting decentralised agent interaction and interface exchange.

5.1 Deterministic Factor Partitioning

To ensure reproducibility and avoid double counting, decentralised correctness depends on deterministic factor ownership.

Key-to-Robot Mapping A global `robot_map` is built once from JRL initialisation blocks using `build_key_robot_map`. All agents share this map to attribute pose keys to robots consistently.

Inter-Robot Ownership Rule For an inter-robot between-factor whose endpoints belong to robots r and s , ownership is assigned deterministically:

$$\text{owner_robot} = \min(r, s).$$

Only the owner inserts that inter-robot between-factor into its local graph. The non-owner couples to the remote endpoint only through separator priors derived from received interface summaries. ROS 2 streaming affects transport only; it does not change the factor classification or ownership logic.

5.2 Solver: Local Optimisation and Remote-Prior Refresh

Each agent maintains a persistent local iSAM2 instance (`c_slam_decentral/agents.py`). Static factors (dataset priors, local odometry, local loop closures, and owned inter-robot between-factors) are inserted once. On each coordination round, the agent refreshes time-varying separator priors from the most recently received interface summaries, performs an iSAM2 update, and then summarises its separators for outbound communication.

Algorithm 2 captures the round-level procedure.

Algorithm 2 Agent round: refresh remote priors, solve locally, and emit separator summaries

Require: agent r , latest remote summaries, round index k

Ensure: updated local estimate \hat{X}_r and outgoing interface messages \mathcal{I}_{out}

- 1: Insert static factors into persistent iSAM2 (once)
 - 2: Build remote separator priors from latest interface messages
 - 3: Remove previous-round remote priors using `removeFactorIndices` (if supported)
 - 4: `iSAM2.UPDATE(remote-prior batch graph, remote-prior batch values)`
 - 5: **if** update fails (indeterminate/duplicate-key) **then**
 - 6: Rebuild iSAM2 from scratch using static factors + owned inter-robot factors + fresh remote priors
 - 7: **end if**
 - 8: $\hat{X}_r \leftarrow \text{iSAM2.CALCULATEESTIMATE}()$; emit optimisation KPI events for this round
 - 9: Compute marginals for communicated separator keys
 - 10: $\mathcal{I}_{out} \leftarrow \text{MAKEINTERFACEMSGS}(r, \hat{X}_r, \text{marginals})$
 - 11: **return** $\hat{X}_r, \mathcal{I}_{out}$
-

Remote-Prior Semantics Remote separator priors are treated as time-varying “summary constraints” and are replaced each round to avoid accumulat-

ing stale priors. If marginal extraction fails for a separator, the implementation falls back to a conservative diagonal covariance to keep communication and optimisation progressing.

5.3 Interface Exchange and Coordination Loop

Agents exchange separator summaries via a peer bus (`c_slam_decentral/communication.py`). Each agent runs in its own OS process via `c_slam_decentral/mp_runner.py`. Synchronisation is purely message-based (no global scheduler). Each agent iterates a fixed maximum number of rounds and may stop early based on a stability condition.

Algorithm 3 describes the per-agent coordination loop.

Algorithm 3 Per-agent distributed coordination loop

Require: agent r , max rounds R , tolerances (τ_t, τ_R) , stable rounds S
Ensure: final estimate \hat{X}_r and convergence flag

- 1: Ingest factors for r from `<factor_prefix>/<robot_id>`
- 2: Initialise peer bus and perform round $k=0$
- 3: **for** $k = 1$ **to** R **do**
- 4: Drain inbox; apply latest summaries as remote priors
- 5: Perform AGENTROUND and publish fresh interface summaries
- 6: Update stability counter from interface deltas; stop if stable for S consecutive rounds
- 7: **end for**

Convergence monitoring After each round, an agent compares successive outbound interface states by the maximum translation delta Δt_{\max} and maximum rotation delta ΔR_{\max} . Convergence is declared when both fall below thresholds for S consecutive rounds (`--ddf-convergence`, `--ddf-rot-convergence`, `--ddf-stable-rounds`). These checks are local to each agent and may trigger at different round indices due to asynchronous message arrivals. Here, Δt_{\max} is computed as the Euclidean norm of the translation difference, and ΔR_{\max} as the angle (in radians) of the relative rotation between successive quaternion estimates. Independently of this early-stopping rule, each agent enforces a hard maximum of R rounds to bound runtime and ensure experimental reproducibility. A run is marked *converged* only if the stability condition is satisfied before reaching the round budget; otherwise it is treated as not converged within the allocated coordination horizon.

5.4 Communication Transport and Message Schema

Interface summaries are encoded as JSON bytes (`c_slam_ros2/interface_msg.py`) and exchanged via per-receiver ROS 2 topics. For run isolation, the runner appends a unique `run_<id>` namespace to the interface prefix, so agents

publish on `<iface_prefix>/run_<id>/<receiver_id>` and subscribe only to their own receiver topic.

Each interface message contains: `sender`, `receiver`, `key`, `stamp`, pose mean (`translation`, `rotation`), covariance (`covariance`), and the consensus `iteration`. Latency and bandwidth are measured from serialised ROS 2 message payload sizes at publish time (excluding DDS/RTPS and lower-layer overhead); analytical byte counts are not used for evaluation because JSON overhead and string keys are significant and run-dependent.

6 Network Impairment and Quality-of-Service Implementation

This section describes how the network conditions from the methodology are realised: impairments are injected at the message level inside the ROS 2 publish paths and are applied consistently across architectures for any chosen impairment *application mode*.

6.1 Impairment Injection Points

Impairments are applied in three places:

1. **Factor publisher** (`tools/ros2_factor_publisher.py`): drops/throttles factor batches before publication.
2. **Decentralised peer bus** (`c_slam_decentral/communication.py`): applies the same impairment rules to interface messages.
3. **Central downlink** (`c_slam_ros2/map_broadcast.py`): applies the same rules to optional map updates.

All impairment logic is encapsulated in `ImpairmentPolicy` (`c_slam_ros2/impair.py`) so that a single JSON profile can be injected into all processes by the orchestrator.

6.2 Impairment Semantics Relevant to KPIs

The impairment policy combines message drop and delay mechanisms. The details matter because they change how latency and delivery-rate KPIs should be interpreted:

- **Random loss and burst loss** drop messages before they enter the ROS 2 transport.
- **Blackouts** disable send and/or receive for a specified robot over an interval.
- **Bandwidth caps** implement a token bucket that *delays* messages (sleep) to enforce an average throughput, rather than dropping by default.

- **Warmup behaviour** can disable random/burst drops at startup to avoid immediate underconstraint, while still applying bandwidth caps and blackouts.

6.3 Impairment Configuration

Impairment profiles are specified as JSON and loaded via `C_SLAM_IMPAIR` (inline) or `C_SLAM_IMPAIR_FILE`. The orchestrator injects the same profile into all processes to ensure that both architectures experience identical impairment settings in comparative runs.

```
{
  "seed": 42,
  "random_loss_p": 0.05,
  "random_warmup_messages": 50,
  "burst_period_s": 10.0,
  "burst_duration_s": 1.0,
  "blackouts": [
    {"rid": "r0", "start_s": 30.0, "end_s": 35.0, "mode": "sender"}
  ],
  "bw_caps_mbps": {"default": 2.0, "r1": 1.0},
  "warmup_s": 5.0
}
```

6.4 Quality-of-Service Configuration

QoS policies (reliability, durability, history depth) are applied consistently to all relevant topics. Controlled sweeps over QoS use `--qos-reliability`, `--qos-durability`, and `--qos-depth`, and the orchestrator forwards the same settings to publisher and solver processes. The baseline profile (`c_slam_ros2/qos.py`) is:

- Reliability: RELIABLE
- Durability: VOLATILE
- History: KEEP_LAST with depth 20

6.5 Team Size Scaling and Namespace Isolation

Team size N is determined by the dataset. Factor batches use `<factor_prefix>/<robot_id>` with completion markers on `<factor_prefix>/control`. The decentralised peer bus uses per-receiver topics `<iface_prefix>/run_<id>/<robot_id>`. Sweep runs additionally use `ROS_DOMAIN_ID` isolation to avoid collisions between repeated or concurrent runs.

7 Performance Metrics Instrumentation

This section describes how the KPIs defined in the methodology are instrumented in software. The guiding principle is that both architectures emit comparable event streams so that KPI derivation can be performed offline in a backend-agnostic way where possible.

7.1 Event Logging and Derivation Pipeline

All KPI-related measurements are recorded as timestamped events and exported to JSON files under `kpi_metrics/`. The system logs:

- **ingest events** (factor reception and attribution to solver batches/rounds),
- **solver events** (`optimization_start/optimization_end` for central batches; per-round solve events for decentralised agents),
- **communication events** (publish sizes for factor batches, interface messages, and optional downlink/ACKs),
- **resource samples** (CPU and memory time series).

Derived KPIs (e.g., stabilisation-based correction times) are computed offline from these event logs by `tools/kpi_derive.py` and written to `kpi_metrics/derived_kpis.json`.

7.2 Latency Instrumentation

Factor-to-solve attribution. Each ingested factor is assigned to a solver batch (centralised) or to a local optimisation round (decentralised) using the internal batch/round indices emitted by the solver loop. This attribution is what enables ingest-to-optimisation and settling-time KPIs without re-running optimisation offline.

Timestamps. Publisher timestamps (`send_ts_mono`, `send_ts_wall`) are carried in the message payload and recorded at reception. Solver timestamps are recorded at optimisation boundaries in the backend processes. This allows separating transport effects (delays/loss) from solver scheduling effects (batching and compute time). From these event pairs, the evaluation derives explicit latency components, including publish-to-receive $\Delta t_{\text{pub} \rightarrow \text{recv}} = t_{\text{recv}} - t_{\text{send}}$, ingest-to-optimisation-start $\Delta t_{\text{ingest} \rightarrow \text{opt}} = t_{\text{optimization_start}} - t_{\text{ingest}}$, optimisation duration $\Delta t_{\text{solve}} = t_{\text{optimization_end}} - t_{\text{optimization_start}}$, and settling time as the interval from factor ingestion to the first subsequent optimisation-end event after which estimate changes remain below a stability threshold.

7.3 Accuracy Instrumentation

Trajectory accuracy is computed from exported estimates using `c_slam_common/metrics.py`. The implementation performs per-robot alignment against JRL

ground truth and exports ATE/RPE summaries to `kpi_metrics/estimation_metrics.json`. The implementation fixes the similarity scale to $s = 1$ (rigid alignment) because COSMO-Bench trajectories are metric, and reports per-robot RMSE values for evaluation plots.

7.4 Resource and Bandwidth Instrumentation

CPU and memory. Resource utilisation is sampled using `psutil` (`c_slam_common/resource_monitor.py`). For decentralised runs, the solver aggregates samples across spawned agent PIDs. Summary statistics are exported to `kpi_metrics/resource_profile.json`.

Bandwidth. Bandwidth is computed from measured serialised message sizes at publish time. For centralised runs, uplink is the sum of factor batch payload sizes; downlink is measured when enabled. For decentralised runs, bandwidth is derived from outgoing interface message payload sizes. Aggregated bandwidth statistics are exported to `kpi_metrics/bandwidth_stats.json`. These measurements reflect application-layer payload sizes and therefore represent a lower bound on link utilisation under DDS/RTPS transport.

Delivery rate under impairments. When impairments are enabled, effective delivery rate is derived by combining publisher-side send counters with receiver-side observed counters (and optionally ACKs for central factor batches). Results are exported to `kpi_metrics/robustness_metrics.json`.

Evaluation

This chapter evaluates the C-SLAM backends by instantiating the defined experimental design space $\mathcal{D} = \mathcal{A} \times \mathcal{N} \times \mathcal{Q} \times \mathcal{I}$ in Section 4.1 with a baseline grid (clean network, reference QoS) and targeted sweeps (team size, QoS, impairments).

A comparison between centralised and decentralised backends often comes with default expectations; for example, centralised is accurate but bandwidth-heavy, whereas decentralised is scalable but slower to settle. Rather than foregrounding these obvious points, the discussion emphasises the non-obvious implications that emerge when accuracy, delivery, and semantics-aware timing are interpreted jointly: (i) *what “latency” means* depends on the architecture’s communication semantics, (ii) *dataset regime and constraint structure* can dominate “architecture” effects, and (iii) *robustness* is shaped by both estimator behaviour and software stability under backlog and intermittency.

1 Experimental Protocol

All experiments replay the same dataset-provided constraint stream (Section 5.1) and log metrics with the KPI pipeline described in Section 5.7. The dataset contains two native regimes (*WiFi* and *ProRadio*). These are treated as environmental conditions, not tunable parameters. Each configuration is repeated five times ($R = 5$), and tables report repeat means with 95% confidence intervals where shown.

The baseline uses the reference ROS 2 QoS profile RV-20 (reliable, volatile, depth 20). The QoS sweep varies only Q at a fixed team size and regime. The impairment sweep varies only \mathcal{I} using the mechanism defined in Section 5.6; in the runs reported here, impairments are applied at *factor publication* (publisher-impaired mode), i.e., they primarily constrain *constraint availability* rather than directly impairing decentralised peer exchange or centralised downlink.

1.1 Timing Semantics

Timing metrics are reported in a way that respects architectural semantics (Section 4.5.1). The centralised backend exposes a clear dissemination event: the server broadcasts a global update. We therefore report t_{global} as a *dissemination-tail proxy* (time until the last broadcast observed in the run). In addition, “Corr. p95” reports loop-closure correction latency (p95), i.e., the time from loop-closure ingestion until that correction is stably reflected.

The decentralised backend has no single broadcast event. For decentralised runs, “Corr. p95” refers to interface correction stabilisation latency (p95) and should be read as a *correction-propagation proxy*. These two “Corr. p95” numbers quantify different pipeline stages and must not be compared as if they were the same definition of convergence.

When a backend-agnostic convergence statement is required, the chapter uses the stabilisation proxy $T_{\text{conv}}^{(\text{team})} = \max_r T_{\text{conv}}^{(r)}$ from Section 4.5.1. Importantly, event-style proxies (such as t_{global} or “Iface p95”) can move in counter-intuitive directions under missing constraints: for instance, a blackout may shorten t_{global} simply because fewer factors arrive (less work and fewer broadcasts), even though estimation quality degrades. Throughout this chapter, timing is therefore interpreted jointly with delivery and accuracy.

2 Results Overview

Across all studies, three patterns recur and shape the interpretation.

First, the dominant “latency” in these backends is not the solver update duration (which is short in both cases), but the *time to a team-visible stable correction*. Centralised runs typically settle rapidly once a broadcast arrives (very small $T_{\text{conv}}^{(\text{team})}$), so perceived delay is largely driven by *when* broadcasts occur. Decentralised runs often show shorter optimisation steps but longer tails for interface-level stabilisation, so perceived delay is dominated by *propagation and settling* rather than raw compute.

Second, the dataset regime interacts strongly with architecture: the WiFi and ProRadio sequences differ in their constraint-stream characteristics, and this can dominate scalability trends. In particular, the decentralised backend behaves qualitatively differently at r5 under WiFi vs. ProRadio, despite identical algorithmic structure and logging.

Third, robustness is not only about estimation under missing factors; it is also about whether the implementation remains stable under backlog, delayed

delivery, or intermittency. The decentralised backend exhibits a repeatable crash signature in several blackout and QoS configurations; these cases are treated as unsupported in the quantitative comparisons and are explicitly analysed as an engineering limitation.

3 Baseline Comparison Under Native Dataset Networking

The baseline establishes a common operating point for both architectures under the dataset-native regimes, using identical factor streams and the same RV-20 QoS profile. It anchors the subsequent team-size, QoS, and impairment sweeps.

3.1 Reference r3 Baseline: Centralised vs. Decentralised

Table 6.1 summarises the r3 baseline. Under both regimes, the centralised backend achieves lower ATE RMSE than the decentralised backend. The more informative observation, however, is *how* this gap coexists with timing and traffic:

(i) *“Faster corrections” is not the same as “faster convergence”*. On WiFi r3, the decentralised “Corr. p95” is smaller than the ProRadio case (4.79 s vs. 14.37 s), yet the shared convergence proxy $T_{\text{conv}}^{(\text{team})}$ remains an order of magnitude larger than the centralised value. This reflects the decentralised system’s correction propagation and settling tail: local updates are frequent and cheap, but the team-level estimate takes longer to stop moving under the stabilisation criterion.

(ii) *“Decentralised uses less bandwidth” is regime-dependent*. On WiFi r3, decentralised traffic is lower than centralised traffic (28.7 vs. 55.0 MiB). On ProRadio r3, decentralised traffic is slightly higher than centralised traffic (32.2 vs. 30.3 MiB). In other words, the decentralised pipeline does not guarantee a bandwidth win; it shifts *where* bytes are spent (peer-to-peer summaries and factor exchange vs. downlink broadcast), and the regime can flip which side is cheaper.

(iii) *Distributed compute does not imply “less compute”*. Figure 6.1 reports a normalised view (CPU-seconds per solver update) to separate “how expensive is an update” from “how many updates happen”.

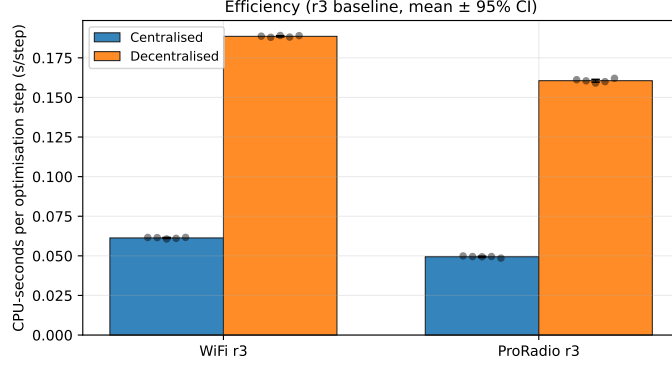


Figure 6.1: Efficiency-normalised metric for the r3 baseline: CPU-seconds per optimisation step. Bars show mean over repeats ($R = 5$); error bars denote 95% CI across repeats, with individual repeats overlaid. Lower values indicate less compute time spent per solver update.

Table 6.1: Baseline comparison for r3 under dataset-native networking. t_{global} is a centralised dissemination proxy (time until last global map broadcast). “Corr. p95” is a correction-latency indicator. For centralised runs it reports loop-closure correction latency (p95), while for decentralised runs it reports interface correction stabilisation latency (p95). These timing indicators are architecture-specific. Although Strict cross-architecture convergence comparisons should use the shared stabilisation proxy $T_{\text{conv}}^{(\text{team})}$ (Section 6.1.1). Link(L): ProRadio(P) & WiFi (W); Backend(B): Centralised(C) & Decentralised(D).

L	B	ATE RMSE (m)	t_{global} (s)	Corr. p95 (s)	$T_{\text{conv}}^{(\text{team})}$ (s)	Traffic (MiB)
P	C	2.164 ± 0.000	11.5	0.096	0.01 ± 0.00	30.3
P	D	3.821 ± 0.072	–	14.37	$\geq 21.33 \pm 5.94$	32.2
W	C	2.270 ± 0.002	16.4	0.126	0.12 ± 0.00	55.0
W	D	3.581 ± 0.035	–	4.79	15.19 ± 3.53	28.7

Traffic split: centralised (up/down) ProRadio 4.1/26.2, WiFi 5.6/49.4; decentralised up=total, down=0.

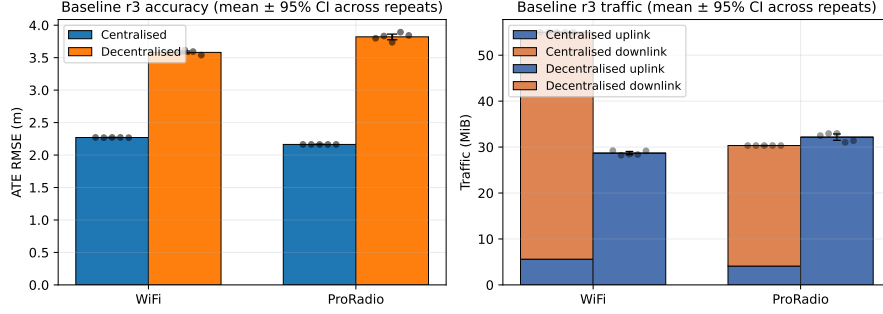


Figure 6.2: Compact summary of the r3 baseline (Table 6.1): ATE RMSE and total traffic with uplink/downlink split. Bars show mean over repeats ($R = 5$); error bars denote 95% CI across repeats.

Detailed resource traces used for diagnosing stability and backlog effects are provided in Appendix Figure A.1.

3.2 Scalability Trend (r3–r5)

Scalability is assessed by repeating the baseline protocol at r4 and r5. Table 6.2 reports the key trend metrics used in the main discussion; the full multi-metric tables (including CPU/RSS and Opt. p95) are provided in Appendix Tables A.1–A.2. For the decentralised backend, Figure 6.5 visualises how final accuracy relates to DDF termination time as team size increases. Diagnostic counts of inter-robot loop closures and interface-update activity are provided in Appendix Table A.3.

(i) *Centralised scalability is driven by dissemination tail, not solver load.* For the centralised backend, total traffic grows with team size primarily because the global map broadcast must serve more robots and the map itself grows as more trajectories contribute constraints. Under WiFi, traffic increases from 55.0 MiB (r3) to 69.0 MiB (r4) and 91.0 MiB (r5). Under ProRadio, the same monotonic growth is observed (30.3 MiB to 74.4 MiB to 86.8 MiB), and t_{global} increases accordingly. Importantly, solver stability (as measured by $T_{\text{conv}}^{(\text{team})}$) remains low across all team sizes, indicating that the centralised optimiser is not the bottleneck.

(ii) *Decentralised scalability is regime-dependent, not uniform.* Under WiFi, decentralised traffic grows from 28.7 MiB (r3) to 41.4 MiB (r4) and 66.9 MiB (r5), and accuracy degrades at r5 (ATE RMSE 6.66 m), suggesting that interface coordination becomes a limiting factor for maintaining global consistency at this team size under the given dataset regime. Under ProRadio, decentralised performance is comparatively stable across team sizes: ATE RMSE is 3.82 m (r3), 2.72 m (r4), and 2.90 m (r5), while traffic grows from 32.2 MiB to 36.1 MiB and 47.4 MiB.

(iii) *Constraint structure mediates scalability more than robot count.* Although WiFi exhibits more inter-robot loop closures than ProRadio (Appendix

Table A.3), WiFi r5 shows worse decentralised accuracy than ProRadio r5. This suggests that the *distribution and timing* of constraints, rather than merely their count, shape how effectively distributed coordination can maintain global consistency. Figure 6.4 complements this trend view by showing the per-repeat ATE–traffic dispersion and the resulting trade-off frontier.

Table 6.2: Condensed baseline scalability trend across team sizes (r3–r5). Link: P=ProRadio, W=WiFi. Backend: C=centralised, D=decentralised. “Time” denotes t_{global} for centralised runs (dissemination proxy) and Corr. p95 for decentralised runs (correction-propagation proxy). $T_{\text{conv}}^{(\text{team})}$ is the shared stabilisation proxy (mean \pm 95% CI; “ \geq ” indicates right-censoring when stabilisation is not observed).

Link	Team	B	Status	ATE RMSE (m)	Time (s)	$T_{\text{conv}}^{(\text{team})}$ (s)	Traffic (MiB)
P	r3	C	ok	2.164 ± 0.000	11.5	0.01 ± 0.00	30.3
P	r4	C	ok	2.526 ± 0.000	21.6	0.12 ± 0.01	74.4
P	r5	C	ok	1.985 ± 0.013	25.0	0.17 ± 0.00	86.8
P	r3	D	ok	3.821 ± 0.072	14.4	$\geq 21.33 \pm 5.94$	32.2
P	r4	D	ok	2.721 ± 0.030	8.3	12.58 ± 1.20	36.1
P	r5	D	ok	2.898 ± 0.028	6.0	$\geq 20.34 \pm 10.35$	47.4
W	r3	C	ok	2.270 ± 0.002	16.4	0.12 ± 0.00	55.0
W	r4	C	ok	2.151 ± 0.000	20.6	0.13 ± 0.00	69.0
W	r5	C	ok	1.885 ± 0.031	26.3	0.15 ± 0.01	91.0
W	r3	D	ok	3.581 ± 0.035	4.8	15.19 ± 3.53	28.7
W	r4	D	ok	3.274 ± 0.030	4.9	11.53 ± 0.45	41.4
W	r5	D	ok	6.656 ± 0.635	6.2	23.85 ± 1.55	66.9

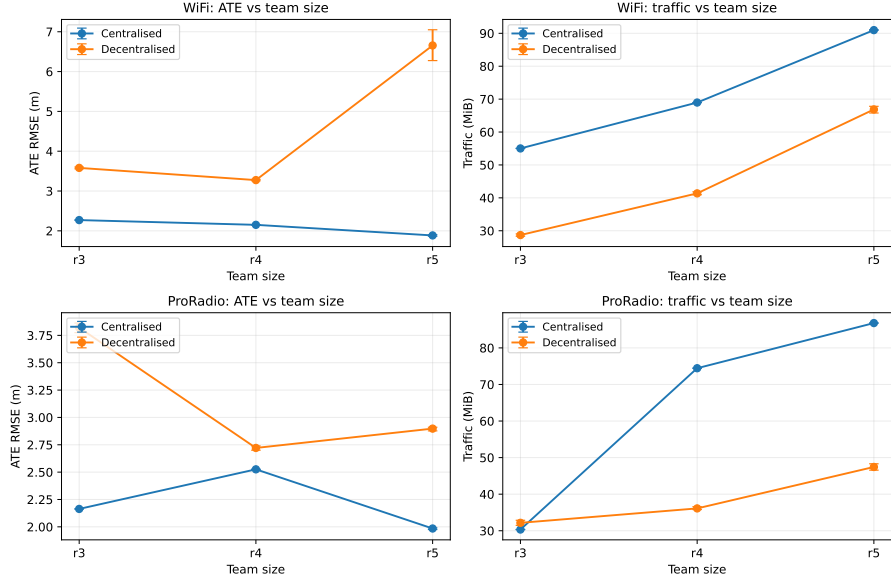


Figure 6.3: Scalability trend in the baseline sweep (Table 6.2): ATE RMSE and traffic vs. team size (r3–r5) for WiFi and ProRadio. Lines show mean over repeats ($R = 5$); error bars denote 95% CI across repeats.

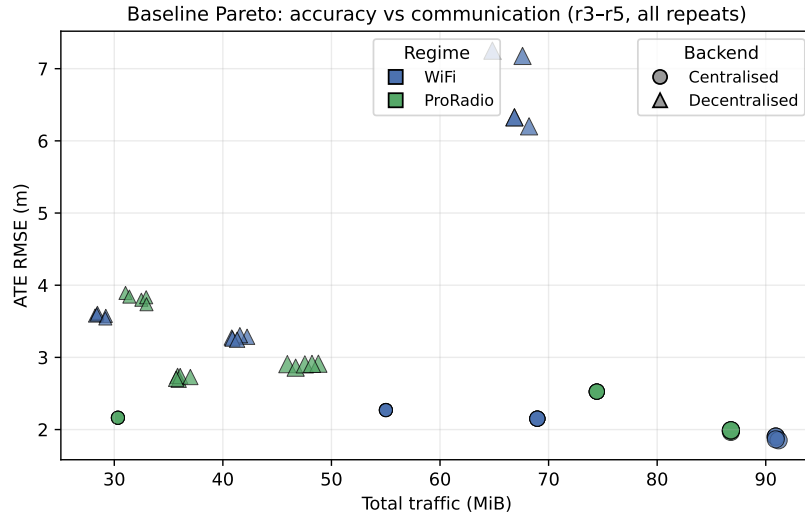


Figure 6.4: Pareto view of baseline trade-offs across team sizes (r3–r5): ATE RMSE vs. total traffic for each successful repeat. This visualisation complements Figure 6.3 by showing per-run dispersion and the accuracy–bandwidth frontier for each architecture.

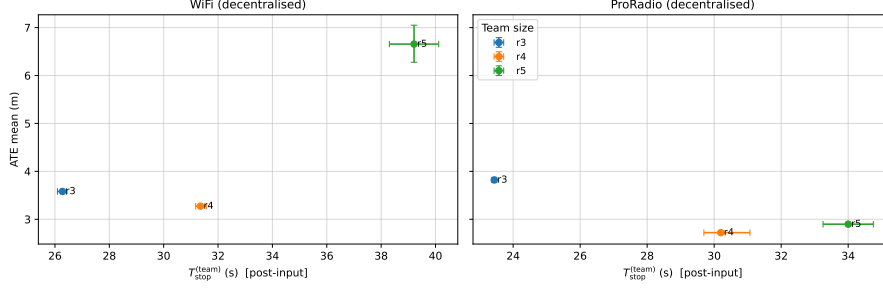


Figure 6.5: Decentralised diagnostic: ATE mean vs. DDF termination time. The x-axis reports $T_{\text{stop}}^{(\text{team})}$ (time from `input_end` until all agents emit `ddf_stop`, team statistic = max over robots). Error bars denote 95% CI over repeats ($R = 5$).

4 Sensitivity to ROS 2 QoS Policy

The QoS study isolates factor \mathcal{Q} by changing only the ROS 2 communication profile while keeping the dataset regime and all other parameters fixed. Two alternate profiles are evaluated: BT-TL-10 and BT-TL-50, which use best-effort reliability and transient-local durability with history depth 10 and 50, respectively. This tests whether relaxing delivery guarantees (best-effort) and modifying buffering (depth) alter dissemination/correction timing, bandwidth, or estimation quality.

(i) *QoS changes do not materially affect centralised operation in the evaluated settings.* Table 6.3 shows the r3 results. For the centralised backend, QoS changes are negligible: ATE and traffic remain unchanged within rounding and t_{global} remains close to the baseline values.

(ii) *QoS primarily influences decentralised bandwidth overhead through buffering depth.* For the decentralised backend, QoS mainly influences bandwidth through history depth. On WiFi r3, BT-TL-50 increases total traffic by about 35.5% relative to BT-TL-10 (33.6 MiB vs. 24.8 MiB). Despite this shift in communication volume, the ATE change remains small, suggesting that the evaluated QoS settings affect buffering overhead more than estimation quality in the nominal regime for the successful WiFi r3 runs.

(iii) *One decentralised configuration is unsupported, bounding QoS generality.* One decentralised QoS configuration (ProRadio r3 under BT-TL-10) does not complete successfully due to an agent crash (Appendix Table A.4). This point is treated as an unsupported configuration in the current implementation and bounds conclusions about QoS sensitivity under that regime/profile pair. Figure 6.6 provides a compact visual summary of these QoS shifts in accuracy and bandwidth relative to baseline.

Table 6.3: QoS sensitivity at r3. Two alternate profiles are evaluated: BT-TL-10 and BT-TL-50, which use best-effort reliability and transient-local durability with history depth 10 and 50, respectively. Baseline (RV-20) results are reported in Table 6.1. “Time” denotes t_{global} for centralised runs and Corr. p95 for decentralised runs. $T_{\text{conv}}^{(\text{team})}$ is the shared stabilisation proxy (mean \pm 95% CI; “ \geq ” indicates right-censoring). Rows marked fail indicate at least one repeat crashed. Link(L) = WiFi(W) & ProRadio(P); QoS profile = BT-TL-10/50

L	B/E	QoS	Status	ATE mean (m)	Traffic (MiB)	Time (s)	$T_{\text{conv}}^{(\text{team})}$ (s)
W	C	10	ok	2.272 ± 0.000	55.0	16.4	0.12 ± 0.00
W	C	50	ok	2.272 ± 0.000	55.0	16.4	0.12 ± 0.00
W	D	10	ok	3.598 ± 0.032	24.8	7.0	17.12 ± 1.45
W	D	50	ok	3.578 ± 0.030	33.6	4.4	13.49 ± 1.08
P	C	10	ok	2.164 ± 0.000	30.3	11.7	0.01 ± 0.00
P	C	50	ok	2.164 ± 0.000	30.3	11.7	0.01 ± 0.00
P	D	10	fail	—	—	—	—
P	D	50	ok	3.859 ± 0.038	37.2	12.7	$\geq 24.54 \pm 1.09$

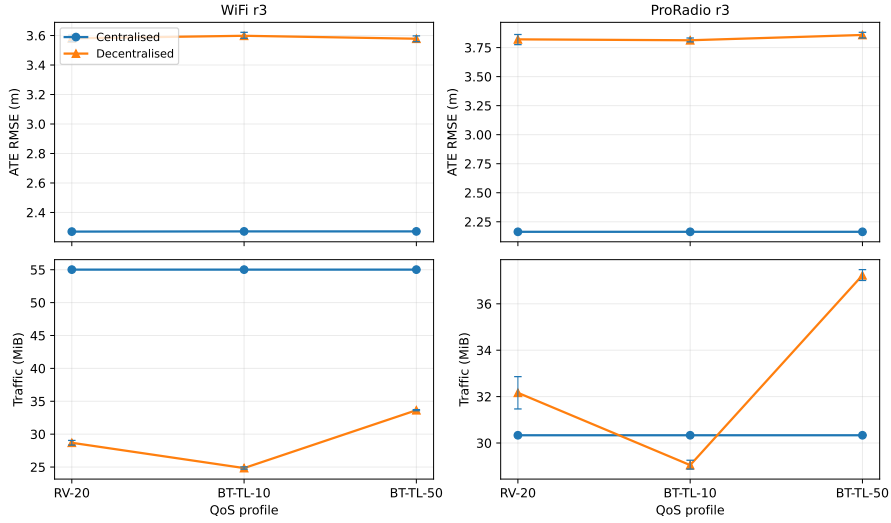


Figure 6.6: QoS sensitivity (r3): summary of how ATE and traffic shift from the baseline profile (RV-20) under BT-TL-10 and BT-TL-50. Lines show mean over repeats ($R = 5$); error bars denote 95% CI across repeats. Failed configurations are marked.

5 Sensitivity to Synthetic Network Impairments

The impairment study varies \mathcal{I} by injecting controlled degradations (Section 5.6). In the reported sweep, impairments are applied to factor publication (uplink/factor-stream degradation). Two impairment families are considered: bandwidth caps (0.25–3.0 Mbps) and periodic blackouts (`blackout_2x`). The evaluation asks not only “does error increase?”, but also *which latency proxy moves, and why*.

5.1 Centralised Backend Under Impairments

For the centralised backend, bandwidth caps on the factor stream have an intuitive effect: they do not change the total information volume (bytes transmitted remains essentially constant), but they delay when batches arrive at the server and therefore extend the map-broadcast tail.

(i) *Severe bandwidth caps inflate dissemination tail without changing total traffic.* With the most restrictive cap (0.25 Mbps), t_{global} increases by factors of $9.6\times$ (WiFi r3) and $10.3\times$ (ProRadio r3), and by $8.0\times$ (WiFi r5) and $8.1\times$ (ProRadio r5). In contrast, caps of 1–3 Mbps are nearly indistinguishable from baseline in both timing and accuracy for the evaluated datasets.

(ii) *Blackouts reduce delivery and can increase ATE, but runs remain stable.* Blackout impairments primarily affect delivery rather than throughput. In the centralised pipeline, publisher-side blackouts reduce factor-stream delivery rates and can change estimation error when critical constraints are missed or delayed, but runs can still complete because the optimiser continues to operate on the received subset of factors and periodically broadcasts the map. Table 6.4 summarises these effects.

Table 6.4: Impact of representative network impairments on the centralised backend. Bandwidth caps of 1–3 Mbps were near-identical to baseline and are omitted here; only the most restrictive cap (0.25 Mbps) is shown. Wr3/5 = WiFi r3/5; Pr3/5 = ProRadio r3/5; bwcap = bwcap 0.25 Mbps; FDM = Factor Delivery Time

Scenario	Impairment	t_{global} (s)	ATE mean (m)	FDM	Traffic (MiB)
Wr3	none	16.4	2.270 ± 0.002	1.000	55.0
Wr3	bwcap	158.1	2.272 ± 0.000	1.000	55.0
Wr3	blackout	10.2	3.192 ± 0.243	0.499	24.3
Wr5	none	26.3	1.885 ± 0.031	1.000	91.0
Wr5	bwcap	210.6	1.902 ± 0.000	1.000	90.9
Wr5	blackout	7.9	1.765 ± 0.058	0.263	22.8
Pr3	none	11.5	2.164 ± 0.000	1.000	30.3
Pr3	bwcap	118.4	2.164 ± 0.000	1.000	30.3
Pr3	blackout	11.7	1.722 ± 0.093	0.742	26.9
Pr5	none	25.0	1.985 ± 0.013	1.000	86.8
Pr5	bwcap	202.0	1.990 ± 0.000	1.000	86.8
Pr5	blackout	21.2	1.652 ± 0.028	0.400	74.1

5.2 Decentralised Backend Under Impairments

For the decentralised backend, impairment response is more regime-dependent. Because the current sweep impairs only factor publication, the peer-to-peer interface exchange is not directly impaired; instead, bandwidth caps and blackouts change the availability and timing of new constraints at each agent.

(i) *Decentralised timing response to bandwidth caps is mixed and depends on regime.* Under WiFi, the evaluated blackout condition leads to agent failures (Appendix Table A.4), while bandwidth caps produce only modest changes in ATE and traffic for the successful runs. Under ProRadio, the strict 0.25 Mbps cap increases interface correction stabilisation latency (p95) and increases ATE at r3, indicating sensitivity to delayed constraint ingestion and slower correction settling under reduced factor throughput.

(ii) *Blackouts trigger stability failures in multiple configurations.* Blackouts are survivable at ProRadio r3 but still trigger agent failures at r4 and r5. WiFi blackouts fail at r3 and r5. These failures indicate that robustness conclusions for decentralised operation under intermittent connectivity are conditional on the stability/supportability of the current implementation. Table 6.5 summarises the decentralised impairment results. Figure 6.7 visualises the normalised slowdown (relative to baseline) for both architectures across impairment conditions.

Table 6.5: Impact of representative network impairments on the decentralised backend (DDF-style). “Iface p95” is interface correction stabilisation latency (p95). Rows marked fail indicate runs where at least one agent crashed. Wr3/5 = WiFi r3/5; Pr3/5 = ProRadio r3/5; bwcap = bwcap 0.25 Mbps; FDM = Factor Delivery Time

Scenario	Impairment	Status	ATE mean (m)	Iface p95 (s)	FDM	Traffic (MiB)
Wr3	none	ok	3.581 \pm 0.035	4.8	0.995	28.7
Wr3	bwcap	ok	3.465 \pm 0.024	9.5	0.989	28.0
Wr3	blackout	fail	–	–	–	–
Wr5	none	ok	6.656 \pm 0.635	6.2	0.998	66.9
Wr5	bwcap	ok	7.603 \pm 0.000	1.6	0.944	64.2
Wr5	blackout	fail	–	–	–	–
Pr3	none	ok	3.821 \pm 0.072	14.4	1.000	32.2
Pr3	bwcap	ok	4.297 \pm 0.044	22.7	0.990	28.9
Pr3	blackout	ok	2.768 \pm 0.031	10.3	0.722	29.7
Pr5	none	ok	2.898 \pm 0.028	6.0	1.000	47.4
Pr5	bwcap	ok	4.876 \pm 0.000	1.4	0.980	41.8
Pr5	blackout	fail	–	–	–	–

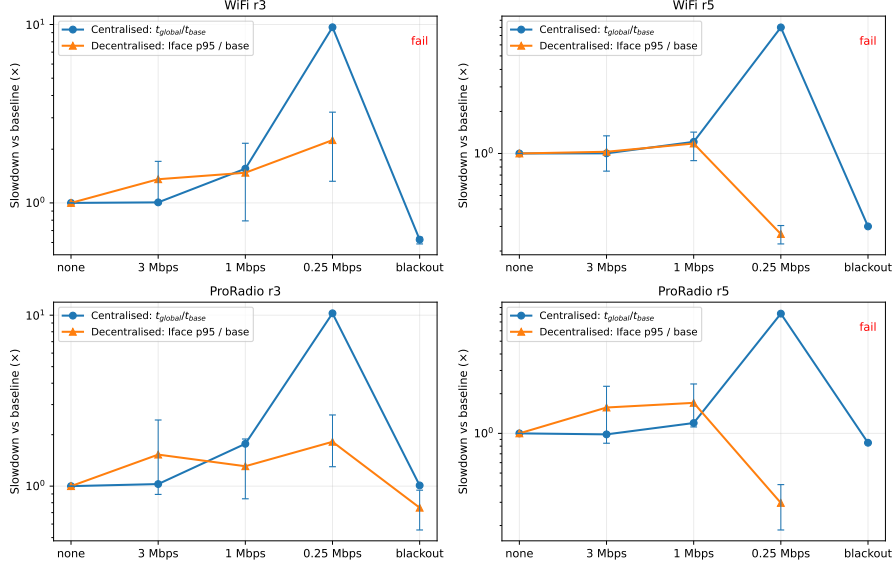


Figure 6.7: Normalised impairment sensitivity (slowdown vs. baseline) under uplink-only (publisher-impaired) degradations (Tables 6.4 and 6.5): centralised $t_{\text{global}}/t_{\text{base}}$ and decentralised Iface p95/ t_{base} . Points show mean over repeats ($R = 5$); error bars denote 95% CI across repeats. Failed runs are marked.

6 Robustness and Unsupported Configurations

Appendix Table A.4 lists configurations that did not complete successfully and were excluded from quantitative comparisons. These outcomes bound the generality of conclusions about QoS and impairment sensitivity for the decentralised backend.

The key robustness takeaway is not merely that “blackouts are hard” (which is expected), but that the observed decentralised failures are triggered by a *specific implementation fragility* under intermittent or delayed delivery illustrates an engineering limitation. Figure 6.8 provides a compact visual summary of which configurations are supported vs. unsupported across the evaluated design space.

Robustness matrix: supported vs unsupported configurations

WiFi r3 C	ok	ok	ok	ok	ok
WiFi r3 D	ok	ok	ok	ok	fail
WiFi r4 C	ok	n/a	n/a	n/a	n/a
WiFi r4 D	ok	n/a	n/a	n/a	n/a
WiFi r5 C	ok	n/a	n/a	ok	ok
WiFi r5 D	ok	n/a	n/a	ok	fail
ProRadio r3 C	ok	ok	ok	ok	ok
ProRadio r3 D	ok	fail	ok	ok	ok
ProRadio r4 C	ok	n/a	n/a	n/a	n/a
ProRadio r4 D	ok	n/a	n/a	n/a	n/a
ProRadio r5 C	ok	n/a	n/a	ok	ok
ProRadio r5 D	ok	n/a	n/a	ok	fail
	Baseline	QoS BT-TL-10	QoS BT-TL-50	Impair 0.25	Impair blackout

Figure 6.8: Robustness matrix summarising which configurations are supported (all repeats ok) vs. unsupported (at least one repeat fails) across the three studies: baseline, QoS, and impairments. Green **ok** indicates all repeats completed successfully; red **fail** indicates at least one repeat failed; grey **n/a** indicates a configuration not evaluated in this study (QoS is evaluated at r3 only; impairments at r3 and r5 only).

6.1 Failure Mode Analysis (decentralised crashes)

Across the failed decentralised runs in Appendix Table A.4, the crash signature is consistent: an out-of-range index in GTSAM iSAM2 factor removal. The exported traces indicate that the exception is raised during `isam.update(..., removeFactorIndices=...)` and is triggered from the decentralised agent update loop. This points to an implementation-level bookkeeping bug (factor-removal indices becoming inconsistent with the internal factor container), rather than an estimator-level divergence.

Resource traces provide supporting context: in representative impairment failures, one or more agents exhibit monotonic RSS growth and high CPU bursts during prolonged intermittency, consistent with queue/backlog accumulation. These conditions plausibly increase the likelihood of stale or invalid factor-removal indices (e.g., through out-of-order updates or mismatched bookkeeping across rounds). In this thesis, these points are treated conservatively as unsupported configurations; claims about decentralised robustness under intermittent connectivity are therefore conditional on resolving this stability issue. Supporting time-series traces are provided in Appendix Figure A.1.

7 Synthesis and Limitations

The evaluation supports four conclusions aligned with the research questions. Each answer identifies the dominant factor that shapes the observed behaviour, rather than merely reporting metric values.

RQ1 (latency and convergence): what dominates “time to a usable update”? Compute time per update is not the dominant contributor in these runs. Centralised updates stabilise rapidly once disseminated (small $T_{\text{conv}}^{(\text{team})}$), so perceived latency is largely driven by dissemination-tail behaviour (t_{global}). Decentralised updates can be compute-cheap but exhibit longer stabilisation tails; perceived latency is therefore dominated by correction propagation and settling.

RQ2 (team size): what actually scales with more robots? For the centralised backend, the key scaling pressure is dissemination payload and the broadcast tail, not solver stabilisation. For the decentralised backend, scaling is regime-dependent: in ProRadio, r4–r5 remain stable with moderate traffic growth, whereas WiFi r5 shows a marked accuracy degradation despite increased interface activity. This indicates that scalability is mediated by constraint structure and coordination dynamics, not solely by team size.

RQ3 (bandwidth/compute under constraints): what changes under uplink degradation? In publisher-impaired mode, centralised runs retain similar traffic volume under bandwidth caps but pay in latency (large increases in t_{global}). Decentralised runs show mixed timing responses under caps because missing/delayed constraints can reduce correction work while harming accuracy; therefore, constraint availability must be considered alongside timing proxies. QoS tuning primarily affects decentralised bandwidth overhead rather than accuracy in nominal conditions.

RQ4 (robustness): where do failures come from? Under intermittent factor availability, centralised runs complete (with reduced delivery and sometimes increased ATE). Several decentralised configurations fail due to a repeatable factor-removal bookkeeping error. This bounds robustness claims for the decentralised backend in the current implementation and highlights a practical requirement for deployment: robustness must include both estimator-level resilience and software-level stability under backlog and out-of-order conditions.

Limitations Two limitations are most relevant. First, the impairment results are scoped to publisher-impaired mode; a complementary sweep that directly impairs decentralised peer exchange and/or centralised downlink would be required to attribute sensitivity to those communication edges. Second, unsupported decentralised configurations limit generality under severe intermittency; the

failure analysis identifies a concrete stability issue that should be resolved to fairly characterise decentralised robustness beyond “does it crash?”.

Note on RPE Relative pose error (RPE) is defined in Section 4.5 as a drift-oriented metric; this chapter focuses on ATE to keep tables compact. RPE values are available in the KPI exports and can be reported in an appendix if required.

Conclusion and Future Works

This research thesis set out to provide a controlled, implementation-grounded comparison of two representative backend paradigms for multi-agent collaborative SLAM. First, it introduces a unified and reproducible ROS 2-based evaluation framework that executes both architectures under a common dataset, factor representation, and KPI instrumentation. This testbed enables fair side-by-side comparisons across latency, accuracy, communication, and resource utilisation. Second, it delivers two concrete backend implementations that represent contrasting architectural choices: a centralised incremental backend that maintains a global factor graph and publishes global updates, and a decentralised DDF-style backend that performs per-agent optimisation and exchanges interface information peer-to-peer. Third, it proposes an impairment-aware experimentation methodology that models packet loss, delay, and bandwidth limitations and applies these consistently across architectures, thereby enabling a systematic assessment of robustness under degraded communication. Finally, the evaluation provides a quantitative characterisation of architectural trade-offs through an empirical analysis spanning baseline operation (3 robots), scalability conditions (4–5 robots), and impaired-network scenarios, reporting ATE/RPE, latency and convergence behaviour, bandwidth consumption, CPU load, and memory footprint.

Taken together, the baseline, scaling, and impairment evaluations provide clear guidance on when each backend is preferable. When global accuracy and predictable convergence dominate system requirements, the centralised backend

is the more suitable choice in the evaluated configurations, particularly as team size grows, where DDF shows both accuracy degradation and the potential for latency inflation due to coordination overhead. When bandwidth is the primary constraint, the decentralised backend is strongly advantageous, consistently achieving large uplink reductions, avoiding downlink entirely, and remaining memory-efficient. Under degraded communication, both systems degrade, but DDF is more sensitive when peer-to-peer corrections are reduced.

Although the presented comparison is intentionally controlled to ensure fairness and interpretability, several limitations remain. The evaluation relies on COSMO-Bench pre-computed factors, which abstracts away frontend-induced effects and therefore does not capture variability introduced by online data association, loop closure discovery dynamics, or keyframe selection policies. In addition, the study contrasts two representative architectural extremes, centralised versus decentralised rather than exploring the broader design spectrum that includes hierarchical, brokered, and opportunistically hybrid systems. The scaling analysis is limited to a maximum of five robots in the reported experiments, and larger teams, heterogeneous compute capabilities, or asymmetric sensing configurations may alter the observed trade-offs. Finally, network impairments are modelled and applied systematically to preserve fairness across architectures; however, real deployments may exhibit more complex dynamics such as mobility-driven link variability, interference, topology changes, and extended partitions that are not fully represented by the impairment models used here.

Building on these findings and limitations, several directions appear promising. One priority is the implementation and evaluation of hybrid and adaptive backends that interpolate between centralised and decentralised operation, for example through temporary brokers, cluster heads, or edge-offload policies that adjust as network conditions change. A second direction is to improve distributed optimisation by investigating interface scheduling, prioritised constraint exchange, and more robust distributed solvers, with the goal of reducing DDF sensitivity to missing peer updates and mitigating coordination overhead at larger team sizes. Third, reintroducing online frontends or comparing multiple frontend variants would enable an end-to-end evaluation of how backend architectural choices interact with real-time loop closure discovery, outlier handling, and sensor heterogeneity. Finally, moving from single-machine multi-process emulation to multi-host deployment across physical machines would provide a more faithful representation of timing, contention, and network effects that arise in realistic multi-robot systems.

Bibliography

- [1] Cesar Cadena, Luca Carlone, Henry Carrillo, Yasir Latif, Davide Scaramuzza, José Neira, Ian Reid, and John J. Leonard. Past, present, and future of simultaneous localization and mapping: Toward the robust-perception age. *IEEE Transactions on Robotics*, 32(6):1309–1332, 2016.
- [2] Yun Chang, Kamak Ebadi, Christopher E Denniston, Muhammad Fadhil Ginting, Antoni Rosinol, Andrzej Reinke, Matteo Palieri, Jingnan Shi, Arghya Chatterjee, Benjamin Morrell, et al. Lamp 2.0: A robust multi-robot slam system for operation in challenging large-scale underground environments. *IEEE Robotics and Automation Letters*, 7(4):9175–9182, 2022.
- [3] Yun Chang, Yulun Tian, Jonathan P How, and Luca Carlone. Kimera-multi: a system for distributed multi-robot metric-semantic simultaneous localization and mapping. In *2021 IEEE International Conference on Robotics and Automation (ICRA)*, pages 11210–11218. IEEE, 2021.
- [4] Weifeng Chen, Xiyang Wang, Shanping Gao, Guangtao Shang, Chengjun Zhou, Zhenxiong Li, Chonghui Xu, and Kai Hu. Overview of multi-robot collaborative slam from the perspective of data fusion. *Machines*, 11(6):653, 2023.
- [5] Ying Chen, Hazer Inaltekin, and Maria Gorlatova. Adaptslam: Edge-assisted adaptive slam with resource constraints via uncertainty minimization. In *IEEE INFOCOM 2023-IEEE Conference on computer communications*, pages 1–10. IEEE, 2023.
- [6] Titus Cieslewski, Siddharth Choudhary, and Davide Scaramuzza. Data-efficient decentralized visual slam. In *2018 IEEE international conference on robotics and automation (ICRA)*, pages 2466–2473. IEEE, 2018.
- [7] Alexander Cunningham, Vadim Indelman, and Frank Dellaert. Ddf-sam 2.0: Consistent distributed smoothing and mapping. In *2013 IEEE international conference on robotics and automation*, pages 5220–5227. IEEE, 2013.

- [8] Alexander Cunningham, Manohar Paluri, and Frank Dellaert. Ddf-sam: Fully distributed slam using constrained factor graphs. In *2010 IEEE/RSJ International Conference on Intelligent Robots and Systems*, pages 3025–3030. IEEE, 2010.
- [9] Jeffrey Delmerico, Stefano Mintchev, Alessandro Giusti, Boris Gromov, Kamilo Melo, Tomislav Horvat, Cesar Cadena, Marco Hutter, Auke Ijspeert, Dario Floreano, et al. The current state and future outlook of rescue robotics. *Journal of Field Robotics*, 36(7):1171–1191, 2019.
- [10] Isaac Deutsch, Ming Liu, and Roland Siegwart. A framework for multi-robot pose graph slam. In *2016 IEEE International Conference on Real-time Computing and Robotics (RCAR)*, pages 567–572, 2016.
- [11] Dapeng Feng, Yuhua Qi, Shipeng Zhong, Zhiqiang Chen, Qiming Chen, Hongbo Chen, Jin Wu, and Jun Ma. S3e: A multi-robot multimodal dataset for collaborative slam. *IEEE Robotics and Automation Letters*, 2024.
- [12] Giorgio Grisetti, Rainer Kümmerle, Cyrill Stachniss, and Wolfram Burgard. A tutorial on graph-based slam. *IEEE Intelligent Transportation Systems Magazine*, 2(4):31–43, 2010.
- [13] Ben Grocholsky, James Keller, Vijay Kumar, and George Pappas. Cooperative air and ground surveillance. *IEEE Robotics & Automation Magazine*, 13(3):16–25, 2006.
- [14] Marco Karrer, Patrik Schmuck, and Margarita Chli. Cvi-slam—collaborative visual-inertial slam. *IEEE Robotics and Automation Letters*, 3(4):2762–2769, 2018.
- [15] Navroop Kaur and Aanchal Sharma. Robotics and automation in manufacturing processes. In *Intelligent Manufacturing*, pages 97–109. CRC Press, 2025.
- [16] Pierre-Yves Lajoie and Giovanni Beltrame. Swarm-slam: Sparse decentralized collaborative simultaneous localization and mapping framework for multi-robot systems. *IEEE Robotics and Automation Letters*, 9(1):475–482, 2023.
- [17] Pierre-Yves Lajoie, Benjamin Ramtoula, Yun Chang, Luca Carlone, and Giovanni Beltrame. Door-slam: Distributed, online, and outlier resilient slam for robotic teams. *IEEE Robotics and Automation Letters*, 5(2):1656–1663, 2020.
- [18] Pierre-Yves Lajoie, Benjamin Ramtoula, Fang Wu, and Giovanni Beltrame. Towards collaborative simultaneous localization and mapping: a survey of the current research landscape. *Field Robotics*, 2:971–1000, 2022.

- [19] Daniel McGann, Easton R Potokar, and Michael Kaess. Cosmo-bench: A benchmark for collaborative slam optimization. *arXiv preprint arXiv:2508.16731*, 2025.
- [20] Manthan Patel, Aditya Bandopadhyay, and Aamir Ahmad. Collaborative mapping of archaeological sites using multiple uavs. In *International Conference on Intelligent Autonomous Systems*, pages 54–70. Springer, 2021.
- [21] Manthan Patel, Marco Karrer, Philipp Bänninger, and Margarita Chli. Covins-g: A generic back-end for collaborative visual-inertial slam. *arXiv preprint arXiv:2301.07147*, 2023.
- [22] L. Riazuelo, Javier Civera, and J.M.M. Montiel. C2tam: A cloud framework for cooperative tracking and mapping. *Robotics and Autonomous Systems*, 62(4):401–413, 2014.
- [23] Angela Ribeiro and Jesus Conesa-Muñoz. Multi-robot systems for precision agriculture. In *Innovation in agricultural robotics for precision agriculture: A roadmap for integrating robots in precision agriculture*, pages 151–175. Springer, 2021.
- [24] Patrik Schmuck and Margarita Chli. Ccm-slam: Robust and efficient centralized collaborative monocular simultaneous localization and mapping for robotic teams. *Journal of Field Robotics*, 36(4):763–781, 2019.
- [25] Patrik Schmuck, Thomas Ziegler, Marco Karrer, Jonathan Perraudin, and Margarita Chli. Covins: Visual-inertial slam for centralized collaboration. In *2021 IEEE International Symposium on Mixed and Augmented Reality Adjunct (ISMAR-Adjunct)*, pages 171–176. IEEE, 2021.
- [26] Abdulhamit Sevgi Ostim, Ahmet Murat Kadioğlu Ostim, and Alpaslan Durmuş Ostim. Using of robotic systems in transportation. In *IEEE International Conference on Cognitive Mobility*, pages 458–468. Springer, 2024.
- [27] David Silvera-Tawil. Robotics in healthcare: a survey. *SN Computer Science*, 5(1):189, 2024.
- [28] Jürgen Sturm, Nikolas Engelhard, Felix Endres, Wolfram Burgard, and Daniel Cremers. A benchmark for the evaluation of rgb-d slam systems. In *2012 IEEE/RSJ international conference on intelligent robots and systems*, pages 573–580. IEEE, 2012.
- [29] Hamid Taheri and Zhao Chun Xia. Slam; definition and evolution. *Engineering Applications of Artificial Intelligence*, 97:104032, 2021.
- [30] Yulun Tian, Yun Chang, Fernando Herrera Arias, Carlos Nieto-Granda, Jonathan P How, and Luca Carlone. Kimera-multi: Robust, distributed, dense metric-semantic slam for multi-robot systems. *IEEE Transactions on Robotics*, 38(4), 2022.

- [31] Hao Xu, Peize Liu, Xinyi Chen, and Shaojie Shen. D2 slam: Decentralized and distributed collaborative visual-inertial slam system for aerial swarm. *IEEE Transactions on Robotics*, 40:3445–3464, 2024.
- [32] Shipeng Zhong, Hongbo Chen, Yuhua Qi, Dapeng Feng, Zhiqiang Chen, Jin Wu, Weisong Wen, and Ming Liu. Colrio: Lidar-ranging-inertial centralized state estimation for robotic swarms. In *2024 IEEE International Conference on Robotics and Automation (ICRA)*, pages 3920–3926. IEEE, 2024.
- [33] Shipeng Zhong, Yuhua Qi, Zhiqiang Chen, Jin Wu, Hongbo Chen, and Ming Liu. Dcl-slam: A distributed collaborative lidar slam framework for a robotic swarm. *IEEE sensors journal*, 24(4):4786–4797, 2023.
- [34] Fangcheng Zhu, Yunfan Ren, Longji Yin, Fanze Kong, Qingbo Liu, Ruize Xue, Wenyi Liu, Yixi Cai, Guozheng Lu, Haotian Li, et al. Swarm-lid2: Decentralized, efficient lidar-inertial odometry for uav swarms. *IEEE Transactions on Robotics*, 2024.
- [35] Danping Zou, Ping Tan, and Wenxian Yu. Collaborative visual slam for multiple agents: A brief survey. *Virtual Reality & Intelligent Hardware*, 1(5):461–482, 2019.
- [36] Chenle Zuo, Zhao Feng, and Xiaohui Xiao. Ccmd-slam: Communication-efficient centralized multirobot dense slam with real-time point cloud maintenance. *IEEE Transactions on Instrumentation and Measurement*, 73:1–12, 2024.

A

Appendix

1 Supplementary Evaluation Material

This appendix collects detailed tables and diagnostic plots that support the evaluation chapter (Chapter 6) but are not required for the main narrative flow.

1.1 Full baseline scalability tables (split)

Table A.1: Full baseline scalability trend across team sizes (r3–r5), results view. Link: P=ProRadio, W=WiFi. Backend: C=centralised, D=decentralised. “Time” denotes t_{global} for centralised runs (dissemination proxy) and Corr. p95 for decentralised runs (correction-propagation proxy). The shared convergence proxy $T_{\text{conv}}^{(\text{team})}$ is reported as the comparison anchor (mean \pm 95% CI across repeats; “ \geq ” indicates right-censoring when stabilisation is not observed).

Link	Team	B/E	Status	ATE RMSE (m)	Time (s)	$T_{\text{conv}}^{(\text{team})}$ (s)	Traffic (MiB)
P	r3	C	ok	2.164 ± 0.000	11.5	0.01 ± 0.00	30.3
P	r4	C	ok	2.526 ± 0.000	21.6	0.12 ± 0.01	74.4
P	r5	C	ok	1.985 ± 0.013	25.0	0.17 ± 0.00	86.8
P	r3	D	ok	3.821 ± 0.072	14.4	$\geq 21.33 \pm 5.94$	32.2
P	r4	D	ok	2.721 ± 0.030	8.3	12.58 ± 1.20	36.1
P	r5	D	ok	2.898 ± 0.028	6.0	$\geq 20.34 \pm 10.35$	47.4
W	r3	C	ok	2.270 ± 0.002	16.4	0.12 ± 0.00	55.0
W	r4	C	ok	2.151 ± 0.000	20.6	0.13 ± 0.00	69.0
W	r5	C	ok	1.885 ± 0.031	26.3	0.15 ± 0.01	91.0
W	r3	D	ok	3.581 ± 0.035	4.8	15.19 ± 3.53	28.7
W	r4	D	ok	3.274 ± 0.030	4.9	11.53 ± 0.45	41.4
W	r5	D	ok	6.656 ± 0.635	6.2	23.85 ± 1.55	66.9

Table A.2: Full baseline scalability trend across team sizes (r3–r5), systems view. Opt. p95 is the p95 optimisation-step duration. CPU and RSS are run means (server for centralised; sum over agents for decentralised).

Link	Team	B/E	Status	Opt. p95 (s)	CPU mean (%)	RSS mean (MiB)
P	r3	C	ok	0.030	32.0	123.9
P	r4	C	ok	0.055	52.7	151.4
P	r5	C	ok	0.058	55.8	155.6
P	r3	D	ok	0.020	37.9	108.3
P	r4	D	ok	0.026	35.0	113.7
P	r5	D	ok	0.026	30.4	113.0
W	r3	C	ok	0.043	45.4	136.4
W	r4	C	ok	0.059	51.7	150.8
W	r5	C	ok	0.062	57.2	159.2
W	r3	D	ok	0.026	38.5	113.0
W	r4	D	ok	0.026	35.4	114.2
W	r5	D	ok	0.028	33.2	114.8

1.2 Decentralised diagnostic counts

Table A.3: Decentralised diagnostic counts in the baseline sweep: number of inter-robot loop-closure constraints (Between factors whose endpoint keys belong to different robot symbols) and number of interface correction updates (count of stabilised interface-correction events). Values are mean \pm 95% CI over repeats ($R = 5$).

Link	Team	Inter-robot loop closures (count)	Interface updates (count)
WiFi	r3	390	19510 \pm 1870
WiFi	r4	609	26231 \pm 3223
WiFi	r5	843	34644 \pm 2990
ProRadio	r3	407	22858 \pm 5315
ProRadio	r4	480	24055 \pm 2408
ProRadio	r5	568	25570 \pm 5564

1.3 Unsupported configurations

Table A.4: Runs that did not complete successfully and were excluded from quantitative comparisons.

Scenario	Study	B/E	Condition	Failure mode
proradio-r3-proradio	qos	D	best_effort_tl_d10	agent c crashed
proradio-r4-proradio	impair	D	blackout_2x	agent d crashed
proradio-r5-proradio	impair	D	blackout_2x	agent e crashed
wifi-r3-wifi	impair	D	blackout_2x	agent c crashed
wifi-r3-wifi	impair	D	bwcap_3p0mbps	agent c crashed
wifi-r4-wifi	impair	D	bwcap_0p25mbps	agent b crashed
wifi-r4-wifi	impair	D	blackout_2x	agent d crashed (and b in one repeat)
wifi-r5-wifi	impair	D	blackout_2x	agent e crashed

1.4 Diagnostic figures

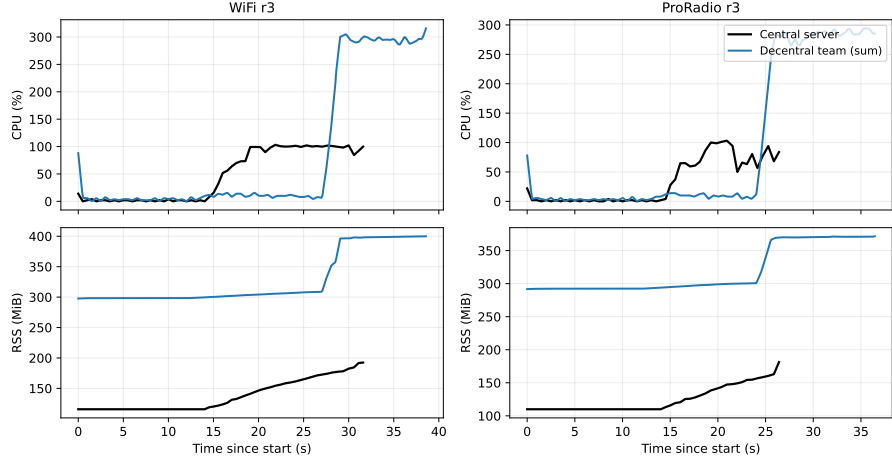


Figure A.1: Resource traces over time for the r3 baseline: centralised CPU/RSS of the server process vs. decentralised CPU/RSS aggregated across agents (sum over per-agent processes). Time is aligned to the start of each run. These traces support diagnosis of stability issues (e.g., monotonic RSS growth preceding agent crashes) and complement the aggregate means reported in Table 6.1.

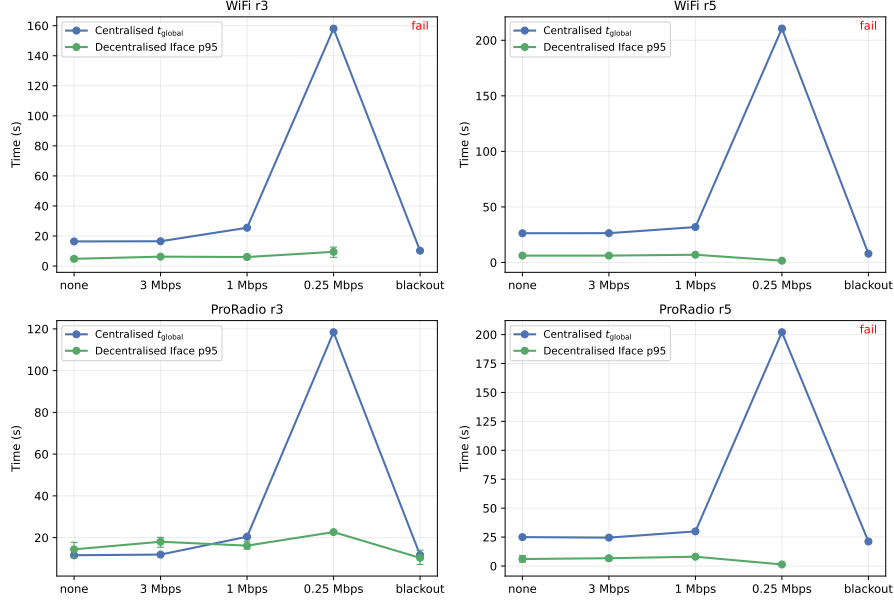


Figure A.2: Absolute timing under impairments (supplementary): centralised t_{global} and decentralised interface stabilisation latency across impairment conditions. This figure complements the normalised slowdown view in Figure 6.7.