NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF
COMPUTER SCIENCE

JOÃO RICARDO BOGALHO BRILHA

BSc in Computer Science and Engineering

MICROBABEL

A MULTI-PROTOCOL APPROACH FOR RESILIENT AND DECENTRALIZED IOT NETWORKS

MASTER IN COMPUTER SCIENCE AND ENGINEERING
SPECIALIZATION IN DISTRIBUTED AND PARALLEL SYSTEMS

NOVA University Lisbon
Draft: *February 1, 2026*

NOVA SCHOOL OF
SCIENCE & TECHNOLOGY

DEPARTMENT OF
COMPUTER SCIENCE

MICROBABEL

A MULTI-PROTOCOL APPROACH FOR RESILIENT AND DECENTRALIZED IOT NETWORKS

JOÃO RICARDO BOGALHO BRILHA

BSc in Computer Science and Engineering

Adviser: João Leitão
Associate Professor, NOVA University Lisbon

MASTER IN COMPUTER SCIENCE AND ENGINEERING
SPECIALIZATION IN DISTRIBUTED AND PARALLEL SYSTEMS

NOVA University Lisbon
Draft: *February 1, 2026*

Abstract

Regardless of the language in which the dissertation is written, usually there are at least two abstracts: one abstract in the same language as the main text, and another abstract in some other language.

The abstracts' order varies with the school. If your school has specific regulations concerning the abstracts' order, the NOVAthesis LaTeX (novathesis) (LaTeX) template will respect them. Otherwise, the default rule in the novathesis template is to have in first place the abstract in *the same language as main text*, and then the abstract in *the other language*. For example, if the dissertation is written in Portuguese, the abstracts' order will be first Portuguese and then English, followed by the main text in Portuguese. If the dissertation is written in English, the abstracts' order will be first English and then Portuguese, followed by the main text in English. However, this order can be customized by adding one of the following to the file `5_packages.tex`.

```
\ntsetup{abstractorder={<LANG_1>,...,<LANG_N>}}
\ntsetup{abstractorder={<MAIN_LANG>={<LANG_1>,...,<LANG_N>}}}
```

For example, for a main document written in German with abstracts written in German, English and Italian (by this order) use:

```
\ntsetup{abstractorder={de={de,en,it}}}
```

Concerning its contents, the abstracts should not exceed one page and may answer the following questions (it is essential to adapt to the usual practices of your scientific area):

1. What is the problem?
2. Why is this problem interesting/challenging?
3. What is the proposed approach/solution/contribution?
4. What results (implications/consequences) from the solution?

Keywords: One keyword · Another keyword · Yet another keyword · One keyword more · The last keyword

Resumo

Independentemente da língua em que a dissertação está escrita, geralmente esta contém pelo menos dois resumos: um resumo na mesma língua do texto principal e outro resumo numa outra língua.

A ordem dos resumos varia de acordo com a escola. Se a sua escola tiver regulamentos específicos sobre a ordem dos resumos, o template (LaTeX) novathesis irá respeitá-los. Caso contrário, a regra padrão no template novathesis é ter em primeiro lugar o resumo *no mesmo idioma do texto principal* e depois o resumo *no outro idioma*. Por exemplo, se a dissertação for escrita em português, a ordem dos resumos será primeiro o português e depois o inglês, seguido do texto principal em português. Se a dissertação for escrita em inglês, a ordem dos resumos será primeiro em inglês e depois em português, seguida do texto principal em inglês. No entanto, esse pedido pode ser personalizado adicionando um dos seguintes ao arquivo `5_packages.tex`.

```
\abstractorder(<MAIN_LANG>):={<LANG_1>,...,<LANG_N>}
```

Por exemplo, para um documento escrito em Alemão com resumos em Alemão, Inglês e Italiano (por esta ordem), pode usar-se:

```
\ntsetup{abstractorder={de={de,en,it}}}
```

Relativamente ao seu conteúdo, os resumos não devem ultrapassar uma página e frequentemente tentam responder às seguintes questões (é imprescindível a adaptação às práticas habituais da sua área científica):

1. Qual é o problema?
2. Porque é que é um problema interessante/desafiante?
3. Qual é a proposta de abordagem/solução?
4. Quais são as consequências/resultados da solução proposta?

Palavras-chave: Primeira palavra-chave · Outra palavra-chave · Mais uma palavra-chave · A última palavra-chave

Contents

Glossary vii

Acronyms viii

1 Introduction 1

2 Related Work 6

2.1 Wireless Communication Technologies . . . 8
2.1.1 Application-Layer Protocols . . . 8
2.1.2 Physical- and Link-Layer Technologies . . . 8
2.1.3 Discussion . . . 9
2.2 Peer-to-peer (P2P) Mesh Networking and Topology Management . . . 10
2.2.1 Peer Discovery and Topology Maintenance . . . 10
2.2.2 Routing in Partitioned and Intermittently Connected Networks . . . 12
2.2.3 Limitations of Centralized Coordination . . . 12
2.2.4 Discussion . . . 13
2.3 Multi-Protocol Communication and Adaptive Protocol Selection . . . 14
2.3.1 Multi-Channel Resilience Architectures . . . 14
2.3.2 Protocol Heterogeneity on Embedded Platforms . . . 15
2.3.3 Runtime Adaptive Protocol Selection . . . 16
2.3.4 Discussion . . . 16
2.4 Decentralized Synchronization and Time Coordination . . . 16
2.4.1 Master-Based Synchronization as Counterexample . . . 17
2.4.2 Gossip-Based Masterless Synchronization . . . 17
2.4.3 Coordination Through Passive View Maintenance . . . 18
2.4.4 Discussion . . . 18
2.5 Lightweight Data Compression and Energy Efficiency . . . 19
2.5.1 Prediction-Based Data Reduction . . . 19
2.5.2 Lightweight Compression . . . 20
2.5.3 Discussion . . . 22

2.6 Summary . . . 23

3 Solution Architecture **24**
3.1 System Requirements . . . 24
3.1.1 Overview of μ-**Babel**'s Approach . . . 24
3.2 Operating Conditions and Failure Modes . . . 25
3.2.1 Target Operating Environments . . . 25
3.2.2 Expected Failure Modes . . . 26
3.3 Hardware Platform . . . 26
3.3.1 Class-1: Battery-Powered Sensor Nodes . . . 27
3.3.2 Class-2: Optional Aggregation Nodes . . . 28
3.4 Software Stack and Development Environment . . . 29
3.4.1 Operating System Layer . . . 29
3.4.2 Platform SDKs and Hardware Abstractions . . . 29
3.4.3 Concurrency Model . . . 30
3.5 System Model . . . 30
3.5.1 Core Components Layer . . . 30
3.5.2 Discovery & Topology Layer . . . 32
3.5.3 Coordination Layer . . . 33
3.5.4 Data Management Layer . . . 33
3.5.5 Component Interactions . . . 34

4 Progress Report and Planning **35**
4.1 Progress Report . . . 35
4.1.1 Use Cases . . . 35
4.2 Planning . . . 37
4.2.1 Validation . . . 37
4.2.2 Task Scheduling . . . 39

Bibliography **41**

List of Figures

3.1 System architecture and component interactions 31

4.1 Proposed timeline 40

List of Tables

2.1 Protocol Characteristics 9
2.2 Related work summary 23

3.1 Class-1 Device Specifications 27
3.2 Supported Communication Protocols 28
3.3 Class-2 Device Specifications 29

Glossary

ESP-NOW Wireless communication protocol developed by Espressif. Enables direct and low-power communication between ESP-family devices without the need for a router. *(pp. 4, 9, 15, 31, 35, 37)*

LoRa From "long range", proprietary radio communication technology *(pp. 3, 4, 9, 13, 14, 19, 31, 35, 37, 38)*

Modbus An application-layer client/server serial communication protocol, widely used in industrial automation systems. *(p. 15)*

MQTT Lightweight publish/subscribe messaging protocol designed for constrained devices and low-bandwidth networks. Previously stood for "Message Queuing Telemetry Transport", but since 2013 it does not stand for anything in particular *(pp. 8, 16, 22, 28, 32)*

Thread Low-power, mesh networking technology based on the IEEE 802.15.4 standard. Increasingly adopted for smart-home applications. *(p. 31)*

μ-Babel MicroBabel *(pp. iv, 2–4, 7–9, 13, 16, 19, 22–25, 27–34)*

ZigBee Low-power, low-data rate wireless communication specification based on the IEEE 802.15.4 standard. Designed for short-range communication in mesh, star or tree network topologies. *(pp. 4, 9, 15, 31, 35)*

Acronyms

AES Advanced Encryption Standard *(pp. 15, 17, 23)*
API Application programming interface *(p. 33)*
ARIMA Autoregressive integrated moving average *(pp. 20, 22)*
AWCT Always Connected Things *(p. 14)*

BLE Bluetooth Low Energy *(pp. 3, 4, 9, 10, 13–15, 19, 31, 33, 35, 37, 38)*

CA Certificate authority *(p. 7)*
CoAP Constrained Application Protocol *(pp. 8, 22, 28, 32)*
CPU Central processing unit *(pp. 30, 32)*
CR Compression ratio *(p. 21)*

DNS Domain Name System *(p. 1)*
DODAG Destination-oriented directed acyclic graph *(p. 11)*
DTN Delay-tolerant networking *(pp. 11, 12, 14)*

ESP-IDF Espressif IoT Development Framework *(p. 33)*

FHSS Frequency-hopping spread spectrum *(pp. 15, 17)*
FIRE Fast Integer REgression *(p. 21)*

GASE Group Authentication Scheme at the Edge *(p. 23)*
GPIO General-purpose input/output *(p. 33)*
GPS Global Positioning System *(p. 19)*

HAL Hardware Abstraction Layer *(p. 33)*
HF High frequency *(p. 14)*
HTTP Hypertext Transfer Protocol *(pp. 12, 22)*
HVAC Heating, ventilation, and air conditioning *(p. 6)*

I²C Inter-Integrated Circuit *(p. 33)*
IoT Internet of Things *(pp. 1–3, 6–12, 14, 19, 22–24, 26, 28, 30, 31)*

KDC Key distribution center *(p. 7)*

LE Low Energy *(p. 31)*
LoRaWAN LoRa Wide Area Network *(pp. 10, 14, 16)*
LPKM Lightweight Polynomial-based Key Management *(p. 23)*
LPWAN Low-Power Wide Area Network *(p. 14)*

MAC Message Authentication Code *(p. 23)*
MDL Minimum description length *(pp. 22, 23)*

novathesis NOVAthesis LaTeX *(pp. i, ii)*
NTP Network Time Protocol *(p. 19)*
NVIS Near Vertical Incidence Skywave *(pp. 14–16)*

OS Operating system *(pp. 32, 33)*
OSPF Open Shortest Path First *(p. 28)*

P2P Peer-to-peer *(pp. iii, 3, 4, 7, 10–13, 15, 27, 29–31)*
PKI Public key infrastructure *(p. 24)*
PLR Packet loss ratio *(p. 11)*
PRNG Pseudorandom number generator *(pp. 15, 17)*
PSRAM Pseudostatic random-access memory *(p. 30)*
PUF Physically Unclonable Function *(pp. 23, 27)*

QoS Quality of Service *(pp. 4, 7, 37)*

RAM Random-access memory *(p. 32)*
RF Radio frequency *(p. 14)*
RGCS Randomized gossip-consensus-based sync *(pp. 17–19, 25, 37)*
RLE Run-length encoding *(pp. 21–23, 38)*
RPL Routing Protocol for Low-Power and Lossy Networks *(pp. 11–14)*
RSSI Received signal strength indicator *(pp. 10, 15)*
RTOS Real-time operating system *(p. 32)*

SBC Single-board computer *(p. 32)*
SDK Software Development Kit *(pp. 32, 33)*

SDN Software-defined networking *(pp. 7, 16, 24)*
SF Spreading Factor *(p. 9)*
SPI Serial Peripheral Interface *(pp. 31, 33)*
SRAM Static random-access memory *(p. 30)*
SSID Service Set Identifier *(p. 9)*

TCP Transmission Control Protocol *(pp. 7, 11, 13, 25)*

UART Universal asynchronous receiver-transmitter *(p. 33)*
UDP User Datagram Protocol *(p. 8)*
UUID Universally unique identifier *(p. 35)*

WSN Wireless Sensor Network *(pp. 11, 14, 15, 17)*

1 Introduction

The proliferation of Internet of Things (IoT) devices has changed how we monitor and interact with physical spaces – from smart homes to industrial facilities – with the global IoT market reaching 18.5 billion connected devices in 2024, and projected to reach 39 billion by 2030 [36], with industry analysts estimating that IoT technologies could unlock between $5.5 and $12.6 trillion in economic value by the same year [10].

However, current IoT systems remain fundamentally dependent on continuous Internet connectivity and centralized cloud infrastructures [44, 18]. The dominant architectural pattern across the industry relies on edge devices collecting data and transmitting it to remote cloud servers for processing, storage, and execution of other control logic. This approach delivers scalability and ease of management but creates a fundamental dependency on network availability.

This centralized model introduces several concerns beyond just connectivity: operational costs for cloud services create economic dependencies on third-party providers and often lead to vendor lock-in, data sovereignty issues arise when sensitive information must travel to and reside on external infrastructure, and system resilience becomes fundamentally tied to the availability of these remote services.

While cloud platforms can benefit IoT deployments by providing additional computational resources and storage capacity, systems that *depend* on constant cloud connectivity sacrifice local autonomy and introduce single points of failure. This cloud-centric model has enabled rapid IoT adoption, but introduces critical vulnerabilities across scenarios where continuous connectivity cannot be guaranteed.

Recent infrastructure failures illustrate the fragility of cloud-dependent architectures: the AWS US-EAST-1 outage in October 2025 [39] disrupted services from banking to smart homes worldwide, demonstrating how a Domain Name System (DNS) configuration error in a *single region* can have cascading effects, leading to global disruptions [37].

Attempts have been made to address these challenges through incremental improvements, such as shifting focus to edge computing in order to reduce latency [18, 44, 32], redundant cloud regions for availability [25, 24], and hybrid architectures that combine local and remote processing [1, 11, 20].

However, these solutions remain fundamentally tied to the assumption of eventual connectivity, and often increase system complexity without eliminating the core dependency. This leads to shortcomings in key areas that demand more fundamental architectural reconsideration:

Limited suitability for hazardous environments: Remote or dangerous locations (industrial sites, disaster-prone areas) require systems that can operate reliably without constant human intervention or stable network infrastructure;

Lack of autonomous operation: Device deployment and operation en masse can be brittle, with little tolerance for individual node failures in the IoT infrastructure, which is naturally susceptible to network failures and intermittent connectivity;

Privacy and data sovereignty concerns: Cloud platforms and other third-parties are oftentimes an unavoidable middle layer between end devices and end users, raising questions about data processing and control.

While these challenges affect both IoT and domotics systems alike – from industrial monitoring to residential automation – they become particularly acute in disaster response and emergency scenarios, where communication infrastructure might become completely unavailable precisely when needed most, rapid deployment with minimal configuration becomes essential, and autonomous operation transitions from desirable to indispensable.

During earthquakes, floods, and natural or human-driven disasters, the need for real-time sensor data (structural integrity, air quality, evacuation routes) and bidirectional communication (threat alerts, user feedback) becomes critical, yet traditional infrastructure often fails first, eliminating both cloud connectivity and local network access points that deployed devices rely on, making support systems for these scenarios unavailable or non-operational.

A pragmatic issue in the context of smart homes is that without cloud connectivity, users cannot interact with their appliances, such that during a Wi-Fi outage smart lights become uncontrollable despite all hardware being physically present. While merely inconvenient domestically, this architectural dependency has critical implications in other contexts and **μ-Babel** aims at addressing such scenarios across different application domains: from residential systems requiring local control, to building monitoring infrastructures that must operate during emergencies, to disaster response networks where autonomous operation becomes essential when traditional infrastructure fails.

While cloud platforms like Amazon Alexa, Google Home, and Microsoft Azure IoT Hub offer convenience for data processing and remote device control, their inherent dependence on continuous Internet connectivity introduces some critical limitations: increased latency from round-trip communication to distant servers [32, 28], reduced availability during network disruptions or cloud outages [42, 24], and poor fault-tolerance when infrastructure fails [3].

These characteristics make cloud-centric architectures fundamentally unsuitable for scenarios that require or prioritize local operation, autonomous behavior, or guaranteed responsiveness during emergencies.

The Babel Ecosystem [16] partially addresses these limitations by enabling devices to operate autonomously without cloud infrastructure, while still supporting cloud integration when connectivity is available and desired.

In this document we propose **μ-Babel**, a lightweight framework targeting embedded platforms (e.g., ESP32, Raspberry Pi Pico) aimed at developing resilient, multi-protocol, and decentralized IoT systems that can operate autonomously during infrastructure failures.

μ-Babel will integrate with the broader Babel Ecosystem, which runs on hardware with greater resources such as full Raspberry Pi boards, desktops/laptops, or servers. This enables a heterogeneous architecture where resource-constrained edge devices can seamlessly interoperate with computational nodes for additional data aggregation, processing, and large-scale coordination.

FIXED: "class-based"

Note on terminology: Throughout this work the term *multi-protocol* is used to refer to heterogeneity in communication technologies (i.e., Bluetooth Low Energy (BLE), LoRa, Wi-Fi, etc.) that have distinct link/physical layers, while *multi-channel* refers to frequency diversity within a single protocol (e.g., 2.4GHz Wi-Fi channels 1-13, LoRa frequency hopping). Multi-protocol operation provides technology diversity for resilience; multi-channel operation provides frequency diversity for both anti-jamming and additional throughput.

Main Research Questions

In this work, we focus on three main research questions:

How can these systems maintain communication and full operation when traditional infrastructure fails?

Conventional deployments of networked devices often heavily rely on Wi-Fi access points, cellular towers, or other centralized infrastructure that can easily become unavailable during disasters, or is altogether unreliable in remote locations.

μ-Babel will address this by leveraging a multi-protocol communication approach, supporting a diverse set of wireless protocols that can operate independently of infrastructure.

By enabling adaptive protocol selection and Peer-to-peer (P2P) mesh formation, devices can establish alternative communication paths when primary channels fail, allowing them to sustain information exchange and system availability.

FIXED: muito ênfase em iot

How can device heterogeneity be leveraged to create and orchestrate these systems? Deployments naturally comprise devices with varying capabilities, from simple resource-constrained nodes, to more capable gateways that might act as aggregators or bridges. Rather than treating this heterogeneity as a limitation, **μ-Babel** will exploit it through capability-aware protocols that allow devices to negotiate roles dynamically via autonomous and automatic discovery.

Resource-rich nodes can serve as data aggregation and processing points, or bridges between a remote deployment and traditional network infrastructure (if desired), while simpler devices focus on sensing and actuation, creating a resilient multi-tier architecture.

How can we achieve (near-)zero-configuration deployment in diverse environments? Straightforward deployments are essential where users cannot perform complex configurations – from smart homes and factories to hazardous areas. **μ-Babel** will provide automatic peer discovery across multiple protocols, self-organizing network formation, and decentralized coordination mechanisms that eliminate the need for pre-configured coordinator nodes or manual network planning. Devices will autonomously establish connectivity with each other, negotiate protocols, and begin operation upon being activated, with minimal configuration effort to enable rapid deployment even in hard-to-reach locations.

FIXED: demasiado foco em emergências

Expected Contributions

We plan to make the following main contributions:

- A decentralized architecture supporting multiple communication protocols (Wi-Fi, BLE, LoRa, ZigBee, ESP-NOW) with adaptive switching based on Quality of Service (QoS) requirements, resource availability and device capabilities;
- A resource-efficient programming framework for embedded platforms that enables autonomous operation without central coordination, providing abstractions for multi-protocol communication, peer discovery, and opportunistic data forwarding;
- A proof-of-concept implementation demonstrating infrastructure- independent operation and automatic disaster-mode failover in a real-world deployment.

Document Structure (Roadmap)

FIXED: added doc structure

The remainder of this document is arranged as follows:

Chapter 2 Reviews widely-used communication protocols and related work in P2P networking, multi-protocol communication, decentralized synchronization and data compression.

Chapter 3 Presents the system requirements, operating conditions, failure modes, hardware and software platforms, and the architectural model of the proposed solution.

Chapter 4 Summarizes the current implementation status and outlines the development and evaluation plan.

might be worth voltar atrás e resumir algo para não ter esta página neste estado

2

Related Work

Internet of Things (IoT) and home automation (domotics) systems share the same technological building blocks (wireless sensors, embedded devices, network communication) but diverge significantly in their operational focus and architectures:

FIXED: added citations

IoT systems [13] typically focus on **data collection and monitoring**, by streaming sensor readings to centralized platforms for analysis and processing, with control functions often being a secondary concern.

Domotics systems [17, 15] instead prioritize **real-time control and actuation** over physical spaces, where responsiveness and local autonomy are paramount for end user experience and privacy.

This distinction gains additional relevance when focusing on resilience requirements: while IoT deployments may tolerate delayed data aggregation and/or temporary connectivity issues in monitoring scenarios, domotics applications (such as emergency lighting control or Heating, ventilation, and air conditioning (HVAC) management) demand immediate local response regardless of network conditions.

Both domains, however, suffer from a common vulnerability when confronted with infrastructure failures during disasters: their predominantly cloud-centric architectures collapse precisely when autonomous operation becomes indispensable.

TOFIX: em vez de "iot systems": smart systems?

The challenges faced by disaster-resilient IoT systems span multiple dimensions:

Infrastructure failures eliminate access points that devices depend on for coordination and operation;

Intermittent connectivity creates network partitions where subgroups of devices must operate autonomously without global state synchronization;

Resource constraints limit the computational, memory and energy budgets available for implementing sophisticated resilience mechanisms in embedded platforms.

These challenges are fundamentally architectural: existing IoT and domotics systems are designed around the assumption of stable infrastructure, treating network partitions and coordinator failures as transient anomalies rather than the norm.

Cloud-centric architectures place control decisions in remote servers, creating dependencies that cannot be satisfied when connectivity fails. Coordinator-based topologies – whether using dedicated gateways, master nodes, Software-defined networking (SDN) controllers, among others – concentrate the risk of failure in single points that can paralyze entire systems if compromised or disconnected. Even ostensibly distributed systems often rely on persistent connections (i.e., Transmission Control Protocol (TCP)) over a single medium, or heavy middleware platforms that exceed the capabilities of embedded devices and, once more, assume some form of infrastructure availability.

The fundamental point of contention is between resilience requirements (autonomy during infrastructure collapse) and resource constraints (limited computation, memory, and energy). Traditional approaches address one at the cost of the other: cloud platforms provide sophisticated coordination but fail during outages; purely local systems avoid external dependencies but struggle with inter-device coordination and protocol heterogeneity with low resource usage.

Disaster-resilient systems require architectural choices that prioritize autonomous Peer-to-peer (P2P) connectivity as the baseline mode of operation rather than having it as a secondary – even exceptional – fallback mechanism.

TOFIX: general statement.. não encontro citations to back this up exactly, sinto que é kind of a given

Naturally, this requires a fundamental rethinking of IoT systems design: how devices discover and communicate with each other using diverse protocol stacks (without depending on centralized control), how they achieve temporal synchronization and coordination without master beacons, how to optimize the usage of limited available resources, and how to maintain secure operation throughout without persistent access to inherently centralized components such as Certificate authorities (CAs) or Key distribution centers (KDCs).

The following sections briefly cover related work across five areas that collectively enable autonomous operation: (Section 2.1) common wireless communication technologies across application, link, and physical layers; (Section 2.2) P2P mesh networking and topology management for autonomous network formation without coordinator dependencies; (Section 2.3) multi-protocol communication and adaptive selection based on Quality of Service (QoS) and device capabilities; (Section 2.4) decentralized synchronization mechanisms for coordinating multi-protocol communication without master nodes; (Section 2.5) lightweight data compression and energy optimization techniques to extend autonomous operation duration and device lifetime.

For each area, we examine how existing approaches handle (or fail to handle) infrastructure failure scenarios, identify architectural assumptions that conflict with disaster-resilient requirements, and position μ-**Babel**'s contributions relative to the current state of the art.

2.1 Wireless Communication Technologies

FIXED: moved protocols para aqui + MQTT and CoAP overview

A core pillar of resilient and decentralized IoT systems is the heterogeneity of available communication capabilities and wireless protocols across different layers of the communication stack. Different technologies naturally expose different trade-offs in terms of infrastructure requirements, range, throughput, discovery mechanisms, and energy consumption – all of which directly influence the design choices around their usage.

There is no single communication protocol that fully addresses the requirements of infrastructure-free and disaster-resilient operation. As such, it is important to establish the capabilities and limitations of each one, as well as their potential for harmonious cooperation within a unified system.

2.1.1 Application-Layer Protocols

At the application layer, two messaging protocols stand out given their wide-spread adoption in IoT deployments:

MQTT: A lightweight publish/subscribe messaging protocol designed for constrained devices and unreliable, low-bandwidth networks. It relies on a centralized *broker* to mediate communication between publishers and subscribers – essential, a server.

Constrained Application Protocol (CoAP) [8, 34]: A RESTful, request/response protocol designed for constrained environments. Operates over User Datagram Protocol (UDP) and assumes the presence of well-known endpoints or proxy servers for resource discovery and access.

Both protocols inherently assume the availability of stable endpoints, and rely on continuous network connectivity in order to operate. These assumptions conflict with the infrastructure-less and failure-prone environments targeted by **μ-Babel**. For this reason, while such protocols are of interest for broader interoperability (see Non-Functional Requirements in Section 3.1.1), they are not a part of the foundational protocols employed by this work.

2.1.2 Physical- and Link-Layer Technologies

This section reviews the physical- and link-layer wireless technologies relevant to resilient and decentralized IoT deployments. While the previous section addressed higher-layer messaging protocols, the technologies discussed here provide the foundational communication stack that determines the behavior of **μ-Babel**.

Table 2.1 details the basic operational characteristics of the wireless communication technologies considered in this work.

get actual power draw values from somewhere

These characteristics drive protocol selection based on deployment context and communication requirements:

Table 2.1: Protocol Characteristics

Protocol	Discovery	Range[1]	Frequency	Data Rate	Power Draw
Wi-Fi	SSID scan	50-100 m	2.4 GHz	1-50 Mbps	High
BLE	Advertising	10-50 m	2.4 GHz	1-2 Mbps	Low
ESP-NOW	Peer list	50-100 m	2.4 GHz	1 Mbps	Very Low
ZigBee	Beacon	10-100 m	2.4 GHz	250 Kbps	Very Low
LoRa	Preamble detection	2-10 km	433/868/915 MHz[2]	0.3-50 Kbps[3]	Low[3]

[1] Outdoor line-of-sight; indoor ranges typically 30-50% lower
[2] Frequency depends on regional regulations
[3] Data rate and power consumption depend on Spreading Factor (SF); higher SF = longer range, lower rate, higher energy cost

Range vs. Throughput Trade-offs: LoRa provides 2-10 km range (with appropriate antenna configurations) but very limited throughput (0.3-50 Kbps), suitable for sparse periodic synchronization between distant partitions. Bluetooth Low Energy (BLE) and ESP-NOW offer moderate range (50-100 m) with higher throughput (1-2 Mbps), appropriate for dense local deployments where multi-hop forwarding is feasible and often required.

Discovery Mechanisms: Different protocols employ distinct peer discovery approaches. BLE advertising enables continuous passive discovery, ESP-NOW requires explicit peer lists, and ZigBee uses beacon-based association. **μ-Babel**'s hybrid discovery mechanism (Section 3.5.2) will accommodate these differences while translating protocol-specific information into standardized peer descriptors.

Energy Considerations: Protocol selection must account for energy costs: Wi-Fi consumes significantly more power than BLE, while LoRa power consumption depends heavily on SF configuration. The data compression mechanisms (Section 3.5.4) interact with protocol selection to minimize overall energy expenditure.

Together, these protocol characteristics illustrate the inherent trade-offs faced by resilient IoT deployments operating without reliable infrastructure. Understanding how range, throughput, discovery mechanisms and energy consumption vary across communication technologies is essential to establish an architecture that can adaptively leverage multiple protocols instead of relying on a single one.

2.1.3 Discussion

The communication technologies highlight that no single protocol can simultaneously optimize for range, throughput, energy efficiency, and autonomy under infrastructure failures. Instead, resilience emerges from the ability to combine the complimentary features of each technology and adapt communication strategies to the specific context and requirements of a given environment.

These observations motivate an approach in which diversity is treated as a first-class requirement, and provide the technological context for the multi-protocol operation and architectural choices discussed in Chapter 3.

2.2 P2P Mesh Networking and Topology Management

Infrastructure-dependent star topologies in which devices communicate via a central coordinator or gateway suffer from a similar downside when compared with single-protocol communication: when that coordinator becomes unavailable or unreachable, the entire network loses connectivity.

This pattern pervades current IoT deployments, from Wi-Fi access point dependencies to LoRa Wide Area Network (LoRaWAN) gateway requirements, and becomes catastrophic in disaster scenarios where central coordinators are most likely to fail first, due to power outages, physical damage, or severe network congestion from increased emergency communications traffic.

FIXED: justify this claim

P2P mesh architectures address this limitation by distributing coordination across all participating nodes, thus eliminating single points of failure. Nonetheless, achieving robust mesh operation requires solving three interconnected challenges: neighbor discovery and establishment of initial connectivity, topology maintenance as nodes join/leave a network, and efficient data routing through multi-hop paths when direct communication becomes impossible or inefficient.

2.2.1 Peer Discovery and Topology Maintenance

As discussed in Section 2.3.2, the Multi-Protocol IoT Gateway implementation [19] demonstrates BLE-based neighbor discovery with Received signal strength indicator (RSSI) measurements for proximity estimation.

While that work focuses on multi-protocol integration, its topology management reveals a limitation of its tree-based approach: the system implements automatic parent reselection when coordinators fail, but the underlying tree structure imposes that nodes can only communicate through their parent-child relationships rather than arbitrary peer connections.

This restriction limits route diversity and resilience, creating dependency chains where a single intermediate node failure can disconnect entire subtrees.

A more sophisticated approach to membership management is presented in HyParView [21], in which each node maintains two distinct partial views for scalability: a small active view (size = fanout + 1) containing nodes with which symmetric links are actively maintained, and a larger passive view serving as a backup pool of potential neighbors that may be promoted to the active view if one of its nodes fails.

The active view is managed reactively, such that nodes are added during join operations and removed upon failure detection, while the passive view is maintained cyclically through periodic shuffle operations that randomly exchange node identifiers between peers.

This hybrid strategy enables remarkable resilience, with the system recovering from 80% node failures with minimal reliability loss (maintaining 95% reliability) and from 50% failures in just 1-2 membership cycles, compared to 60+ cycles required by purely cyclic protocols like Cyclon [41].

HyParView's deterministic flooding approach, by broadcasting messages along the entire active view graph rather than probabilistic neighbor selection, enables fast failure detection since every active link is tested at each broadcast. The symmetric link requirement ensures bidirectionality: if node A can reach node B, then B can reach A, preventing the formation of one-way communication paths that complicate routing.

However, HyParView assumes TCP availability for maintaining persistent connections and using connection failures as implicit failure detectors. Additionally, it assumes that any node can communicate with any other node in its active view, which in practice requires some form of routing infrastructure.

This dependency on full network stack functionality makes direct application to resource-constrained embedded platforms challenging, though the architectural principles of hybrid views and shuffle-based passive view maintenance remain valuable.

The heterogeneous disaster IoT architecture [29] discussed in Section 2.3.1 employs Routing Protocol for Low-Power and Lossy Networks (RPL) for its Wireless Sensor Network (WSN) layer, demonstrating practical routing in resource-constrained disaster scenarios.

Their performance analysis reveals considerable trade-offs: convergence time scales linearly from 7 seconds for 20 nodes to 14.5 seconds for 100 nodes, while Packet loss ratio (PLR) at 10 hops reaches 80% with a 10-second sending interval but becomes acceptable at 20-second intervals.

These measurements highlight the throughput limitations imposed by tree topologies, in that their Instance 1 traffic (human data) must be restricted to 1-hop from the root node to ensure the least possible delay in its delivery, defeating the purpose of multi-hop mesh for critical communications.

The Delay-tolerant networking (DTN) component using mobile drones as data mules provides an alternative path for partitioned networks, with a maximum of 20 nodes per Destination-oriented directed acyclic graph (DODAG) to maintain acceptable convergence times during emergency situations, but this approach trades latency for eventual delivery rather than real-time mesh routing.

These approaches reveal fundamental trade-offs: tree-based protocols like RPL offer structured routing for resource-constrained devices but create single points of failure; fully distributed protocols like HyParView achieve resilience through P2P overlay management

but assume network capabilities impractical for embedded platforms; and hybrid multi-layer approaches demonstrate practical deployment but still face throughput limitations.

FIXED: add take-away for the section + set up narrativo "motivates exploration..."

This gap between resilience requirements and embedded capabilities motivates the exploration of topology management strategies more suitable for autonomous operation, particularly in unstable network conditions.

2.2.2 Routing in Partitioned and Intermittently Connected Networks

When network partitions prevent end-to-end paths, store-and-forward mechanisms enable eventual data delivery. The Bundle Protocol [33] addresses delay-tolerant networking through custody-based retransmission and opportunistic connectivity exploitation. While the protocol specification predates modern IoT deployments and was not designed specifically for resource-constrained devices, its core principles inform contemporary DTN approaches.

The framework presented in [26] implements elastic bandwidth utilization by dynamically adjusting transmission rates based on available connectivity, and supports scheduled, predicted, and opportunistic transmission windows, taking inspiration from the Bundle Protocol.

Data parcels are compressed, encrypted, and bundled before transmission, with a load balancer managing concurrent transfer threads to optimize bandwidth usage during brief connectivity windows.

While their Hypertext Transfer Protocol (HTTP)-based implementation targets cloud-backed IoT deployments, the core concepts of parceling data, maintaining transmission queues, and opportunistic forwarding during connectivity windows translate to P2P scenarios where aggregation nodes become neighbors in a mesh network.

2.2.3 Limitations of Centralized Coordination

The work on Resilient Edge-enabled IoT [9] addresses coordinator failures through dynamic leader election and backup mechanisms within their framework.

Their coordination model divides environments into collaboration areas, with resource-rich edge devices serving as coordinators that allocate tasks to workers under their supervision when problems arise. Workers, in turn, are *active agents* such as robots and IoT devices that reside in a particular environment, detect problems, and notify their coordinators.

When coordinators fail, the system automatically elects backups through adaptive decentralized consensus, providing "gentle degradation" during failures with restoration after recovery. Multiple coordinators operate independently, eliminating single points of failure within the coordination model itself.

This architecture depends fundamentally on edge servers running JVM-based SCAFI middleware (a Scala library) to execute the aggregate programs that specify coordination

behavior, and these heavyweight infrastructure requirements – both the Java runtime environment and resource-rich edge computing nodes – conflict with embedded platform constraints and infrastructure-failure scenarios.

queremos mingle com o Babel... se calhar não entrar por aqui ou tudo bem porque não vamos usar os big-raspis para coordination?

While the aggregate computing paradigm separates concerns (sensing, actuation, communication, coordination), the implementation assumptions make it unsuitable for disaster-resilient systems where edge servers may be the infrastructure that fails. The formal guarantees of self-stabilization and compositional properties come at the cost of persistent computational infrastructure that an embedded-focused deployment cannot provide.

2.2.4 Discussion

Existing peer discovery and topology mainteance approaches demonstrate mechanisms appropriate for distributed operation but not without their limitations for resource-constrained platforms aimed at disaster scenarios.

HyParView's [21] hybrid view architecture (small active, large passive) provides remarkable resilience to node failures, but the assumption of routing and TCP availability and persistent connections is not suitable for embedded devices using connectionless radio technology or operating under intermittent connectivity conditions.

The multi-protocol gateway's [19] BLE-based neighbor discovery works on our targeted hardware but imposes tree topologies with parent-child communication restrictions that limit route diversity and create dependency chains.

RPL routing in the heterogeneous disaster architecture [29] demonstrates practical WSN operation but has significant throughput limitations: 20-second minimum send intervals at 10 hops, and restriction of critical traffic to 1-hop from root nodes.

The store-and-forward mechanism presented in the elastic bandwidth framework [26] enables partition tolerance but targets cloud-backed deployments rather than P2P mesh scenarios.

Coordination frameworks like the one in [9] provide dynamic leader election but depend on JVM-based middleware on resource-rich edge servers.

We have found no work that integrates connectionless neighbor discovery, hybrid view maintenance, true P2P routing (not tree-based), and partition-tolerant store-and-forward on resource-constrained embedded platforms.

μ-**Babel**'s approach (Section 3.5.2) must adapt HyParView's architectural principles to BLE/LoRa connectionless communication while supporting multi-hop routing without depending on coordinators or routing infrastructure.

2.3 Multi-Protocol Communication and Adaptive Protocol Selection

Single-protocol (i.e. Wi-Fi only, BLE only, LoRa only, etc.) communication architectures are susceptible to a fundamental vulnerability: if their chosen medium becomes unavailable or inefficient due to interference, range limitations, or infrastructure failures, the entire system loses connectivity and most times ceases to operate altogether.

This becomes especially grave in disaster scenarios where communication conditions change unpredictably due to factors such as Radio frequency (RF) interference caused by debris, obstacles blocking line-of-sight propagation, or damaged/unavailable access points.

Existing approaches to multi-protocol IoT systems can be divided into three categories: multi-channel resilience architectures that orchestrate communication technologies for emergency scenarios; implementations that show the feasibility of protocol heterogeneity on embedded platforms; and adaptive selection frameworks that switch protocols based on runtime conditions.

2.3.1 Multi-Channel Resilience Architectures

Several disaster-focused systems explicitly address infrastructure failures through the exploitation of protocol diversity.

The Always Connected Things (AWCT) framework [22] orchestrates Low-Power Wide Area Network (LPWAN) on top of LoRaWAN with ad-hoc networks (Bluetooth and Wi-Fi) specifically for standby emergency communication. Their architecture adds three modules to standard IoT devices (Raspberry Pi boards, in their test case): a battery module for power management, a power interrupt handler that triggers emergency mode when the power grid fails, and an ad-hoc bridge that forwards packets between the Bluetooth, Wi-Fi and LPWAN interfaces.

The system leverages dense IoT device deployment to provide emergency coverage, demonstrating that existing infrastructure can still serve a purpose during scenarios where the main power grid suffers issues.

However, AWCT's reliance on centralized LoRaWAN gateways for Internet connectivity creates a single point of failure when those gateways become unreachable.

A more comprehensive heterogeneous approach is presented in [29], which integrates High frequency (HF) radio with Near Vertical Incidence Skywave (NVIS), satellite links, WSNs, and DTN with mobile drones for disaster monitoring. Their system uses RPL [2] with three separate instances to differentiate traffic by priority: human data (voice/text via Bluetooth) receives the highest priority, followed by drone-collected data and finally sensor data.

The NVIS backhaul provides 250 km coverage radius without line-of-sight requirements, offering a cost-effective alternative to satellite communications.

While demonstrating successful real-world validation in Antarctica and urban deployments, the architecture's core NVIS topology with centralized coordination contrasts fully distributed operational requirements. Furthermore, their WSN layer requires a minimum 20-second sending interval to maintain acceptable packet loss rates at 10 hops, which highlights throughput limitations of single-channel tree topologies.

Security-focused multi-channel approaches like MCSC-WoT [6] combine Advanced Encryption Standard (AES) encryption with dynamic channel hopping across 2.4 GHz Wi-Fi channels to defend against jamming and eavesdropping attacks. Their lightweight synchronization mechanism minimizes energy consumption while maintaining security through Frequency-hopping spread spectrum (FHSS) patterns generated via Pseudorandom number generator (PRNG). Nodes that lose synchronization can rejoin by hopping to random channels and waiting for the next synchronization signal.

However, the system depends on a master node broadcasting synchronization signals and uses an initial PRNG seed shared among all nodes, raising questions about scalability and seed distribution mechanisms – particularly how new nodes can acquire seeds if deployed to a network some time after the system is operational.

2.3.2 Protocol Heterogeneity on Embedded Platforms

The work presented in [19] shows the technical feasibility of multi-protocol operation on commodity Wi-Fi/BLE modules by integrating ESP-NOW, ZigBee, and Modbus protocols on devices from the ESP32 family to construct multi-hop, tree-based wireless networks.

In this tree-based architecture, devices organize hierarchically with a central gateway serving as the root. Their implementation uses BLE advertising beacons for neighbor discovery with RSSI measurements (for distance estimation), computing parent selection priorities based on weighted combinations of child count, RSSI values, and hop count in the network tree of a given node. The system also supports automatic parent reselection when the current parent becomes unreachable, making it somewhat resilient to single node failures.

Testing demonstrated successful multi-hop operation up to 5 hops in office environments, but evaluation was limited to linear network topologies. Additionally, this architecture relies on a tree topology with parent/child communication that is centered around a gateway (coordinator) rather than being a P2P mesh, creating a single point of failure at the coordinator node.

2.3.3 Runtime Adaptive Protocol Selection

The MINOS platform [40] exemplifies the limitations of centralized multi-protocol approaches. While providing sophisticated multi-protocol support (CORAL-SDN and Adaptable-RPL) with dynamic protocol deployment and real-time parameter tuning, the system depends fundamentally on a centralized SDN controller, MQTT broker, and web server infrastructure. The architecture's single point of failure means that when the controller becomes unreachable the entire system loses its adaptive capabilities and reverts to static operation at best, or complete failure at worst.

FIXED: removed MDPI paper

2.3.4 Discussion

Existing frameworks demonstrate the technical feasibility of heterogeneous channel/protocol coordination but rely on centralized control mechanisms incompatible with disaster scenarios. AWCT [22] depends on centralized LoRaWAN gateways for Internet connectivity, the heterogeneous disaster architecture [29] uses centralized NVIS coordination with minimum 20-second send intervals at 10 hops, MCSC-WoT [6] requires master nodes broadcasting synchronization signals, the multi-protocol gateway [19] uses tree topologies with gateway coordinators as single points of failure, and MINOS [40] assumes SDN controllers with persistent MQTT/web infrastructure.

We have found no work that addresses fully distributed multi-protocol coordination where nodes can autonomously select and switch protocols without resorting to persistent infrastructure. **μ-Babel**'s approach to decentralized protocol selection is detailed in Section 3.5.1 along with masterless synchronization mechanisms.

2.4 Decentralized Synchronization and Time Coordination

Multi-channel communication strategies, particularly those that employ coordinated channel hopping for security or efficiency reasons, require nodes to maintain synchronized clocks. Without time synchronization, devices cannot agree on when to switch channels, rendering coordinated multi-protocol operation unfeasible.

Traditional master-based synchronization approaches – where a designated coordinator is responsible for communicating timing signals – once more fall into the limitation of having a single point of failure in that same master node: should it fail, the entire network becomes susceptible to losing accurate temporal reference, leading to the collapse of any coordination efforts.

This is a notorious challenge to overcome when designing distributed systems, and it becomes more prominent in the disaster scenarios discussed thus far, where infrastructure failures are likely to eliminate the nodes responsible for time synchronization in a given system.

Achieving robust time synchronization in such situations requires fully decentralized approaches that eliminate central coordinator dependencies while still remaining sufficiently lightweight to be executed on resource-constrained platforms.

2.4.1 Master-Based Synchronization as Counterexample

As discussed in Section 2.3.1, the MCSC-WoT framework [6] demonstrates a security-focused multi-channel hopping approach aimed at embedded platforms, but with a fundamental dependency on a master node broadcasting synchronization signals.

This approach uses a shared PRNG seed to generate channel-hopping sequences, with the coordinator node broadcasting periodic synchronization beacons that participant nodes use to compensate for clock drift. Nodes that lose synchronization altogether can rejoin by hopping to a random channel and waiting for the next coordinator beacon.

While the proposed goals were achieved in terms of minimizing energy expenditure and maintaining FHSS patterns, it inherits the fundamental limitation of all master-based approaches mentioned at the start of Section 2.4. If the master fails or becomes unreachable, participating nodes gradually drift out of synchronization until coordinated channel hopping is no longer possible.

The system's clock drift compensation algorithm demonstrates the feasibility of synchronization on ESP32 platforms, and their measured AES encryption performance validates that lightweight security can coexist with time synchronization on resource-constrained devices. However, the architectural dependency on a central coordinator fundamentally conflicts with infrastructure-independent operation requirements we aim to fulfill.

2.4.2 Gossip-Based Masterless Synchronization

Decentralized time synchronization eliminates coordinator dependencies by having nodes reach consensus on clock values through distributed local interactions. Two complementary approaches demonstrate the viability of gossip-based synchronization for WSNs.

The Randomized gossip-consensus-based sync (RGCS) algorithm proposed in [43] addresses time synchronization in dynamic WSNs through randomized asynchronous gossip. Each node maintains a logical clock (T) composed of rate (α) and offset (β) parameters, which together transform the node's hardware clock (τ) into synchronized logical time.

Rather than requiring fixed communication links between specific node pairs which might be fragile in dynamic topologies, their approach uses Poisson-based randomized link activation where each potential synchronization link activates with intensity λ.

The synchronization process operates through pairwise gossip exchanges: when a link activates, the triggering node sends a Sync-L beacon selecting a triggered neighbor, followed by bidirectional exchange of multivariable messages containing each node's current logical clock parameters $[\alpha, \beta, \tau]$.

The asynchronous randomized timing of these exchanges is advantageous in that collision rates drop to near zero compared to 19-23% for deterministic communication protocols, as independent Poisson intervals make simultaneous transmissions to the same receiver statistically unlikely.

The proposed RGCS employs a converge-to-max criterion rather than average-value consensus. This maximum-based approach achieves finite time convergence significantly faster than average-based protocols that require many iterations to converge under significant clock drift. Offset compensation follows suit, with nodes adjusting β parameters based on the difference between their and the neighbors' logical clocks.

Bounded communication delays are handled through a least-square estimation low-pass filter.This addresses the realistic concern of uplink and downlink delays differing, and avoids the symmetric delay assumptions made by many theoretical protocols. The filter's weighing parameter decreases over time, in order to restrain the negative effects of additive noise in stochastic approximation.

Storage complexity remains O($|N_i|$) per node per iteration (proportional only to the number of neighbors, not network size) thus ensuring scalability. The protocol simultaneously compensates both clock rate and offset, unlike approaches that handle these separately and thus require additional convergence time.

2.4.3 Coordination Through Passive View Maintenance

As discussed in Section 2.2.1, the shuffle-based passive view maintenance presented in HyParView [21] provides a complementary coordination mechanism. While primarily designed for topology management, the periodic shuffle operations in which nodes exchange lists of known peers can also provide a *catalog* of potential synchronization partners beyond their immediate active neighbors.

FIXED: moved NTP and GPS talk to discussion

Nodes performing shuffle exchanges already communicate periodically; these same communication windows can opportunistically carry synchronization messages (piggybacking), reducing protocol overhead. The passive view serves as a pool of potential sync partners, so that when a node's active sync neighbors become unreachable, it can initiate sync exchanges with passive view members, providing added resilience to topology changes without requiring global network knowledge.

2.4.4 Discussion

Master-based synchronization approaches like MCSC-WoT [6] demonstrate feasibility on ESP32 platforms with measured AES encryption coexistence, but create single points of failure when master nodes become unreachable.

RGCS [43] eliminates coordinator dependencies, and provides converge-to-max synchronization with Poisson-based randomized gossip that achieves near-zero collision rates with just O($|N_i|$) storage complexity per node. However, it assumes homogeneous

single-protocol networks where all nodes use the same communication medium with identical energy characteristics and range properties.

HyParView's [21] shuffle-based passive view maintenance provides complementary coordination through periodic peer exchanges that could carry piggybacked sync messages, but was not designed for time synchronization integration. One example would be when a node receives a shuffle message containing peer descriptors (IDs and capabilities): it can evaluate these peers as candidates for time synchronization based on their advertised characteristics – for instance, prioritizing peers with access to Network Time Protocol (NTP), Global Positioning System (GPS) clock synchronization, or simply those maintaining more reliable clock sources.

FIXED: aqui

We have found no work that addresses decentralized time synchronization across heterogeneous multi-protocol networks where different communication technologies (BLE, LoRa, Wi-Fi) have distinct energy costs, range, and reliability properties that warrant protocol-specific gossip rates.

μ-**Babel**'s approach (Section 3.5.3) will strive to extend RGCS to multi-protocol scenarios with per-protocol Poisson processes while preventing protocol selection oscillations during sync events.

2.5 Lightweight Data Compression and Energy Efficiency

Energy constraints present one of the most fundamental limitations in battery-powered IoT deployments, particularly in hazardous deployments where replacements and recharging are impractical, and during disaster scenarios where grid power likely becomes unavailable.

TOFIX: não fiz nada em relação ao flow fraquito desta section porque provavelmente vou compactar muito mais. thoughts?

While multi-protocol communication and decentralized coordination efforts provide resilience, they also introduce increased energy costs through frequent transmissions, protocol adaptation overhead and forwarding costs.

Data transmission dominates energy consumption across wireless technologies: LoRa and SIGFOX can represent up to 99.9% of power consumption for transmission, while even shorter-range technologies like BLE and IEEE 802.15.4 dedicate 85-90% of their energy budget to radio operations [38].

This reality makes data reduction techniques essential for extending operational lifetime in infrastructure-independent scenarios. However, compression itself consumes energy through additional computational work, thus creating an unavoidable tug-of-war between compression overhead and transmission benefits.

2.5.1 Prediction-Based Data Reduction

The Ambrosia protocol [38] demonstrates that lightweight prediction can achieve substantial data reductions on resource-constrained devices. The core idea is that not all sensor

readings need to be transmitted if their values can be accurately predicted at the server (within the bounds of a specified error threshold), provided that both sender and receiver maintain a synchronized "prediction state".

Their approach uses window-based forecasting where the next sample is predicted by adding the average differences between recent previous samples, making it dramatically lighter than sophisticated time-series forecasting techniques like Autoregressive integrated moving average (ARIMA) [35] while still achieving comparable data reduction performance.

The protocol works by sending the first w (window size) true samples collected by the sensor node to the server, and for every sample after that comparing its true value to the one predicted locally; the true sample value is sent to the server only if the absolute difference between it and the prediction is greater than a user or application-specified error threshold δ.

The same simple prediction scheme is used on both ends of this communication, and the predicted value is the one used for further predictions – even on the sensor node that knows the true value, to ensure consistency between the two endpoints.

This design achieves up to 60% data reduction with appropriate δ configuration while maintaining sufficient accuracy for diverse applications: for error-tolerant use cases like anomaly detection, δ values will be higher to allow for reasonable fluctuations, thus enabling more significant data reductions, while error-sensitive applications requiring higher precision will naturally lean towards stricter error thresholds, but still benefitting from some amount of data reduction.

The major takeaway from this work is that the window-based prediction executed 99% faster than ARIMA forecasting in their evaluation, making it viable for streaming sensor data at high rates. The approach achieved 2× battery lifetime improvement in high-traffic scenarios and demonstrated compatibility with extremely constrained devices (livestock ear tags with Atmel 8-bit microcontrollers).

2.5.2 Lightweight Compression

While prediction-based reduction minimizes the number of transmissions, compression techniques are essential to reduce the size of the data that *must* be transmitted.

A few complementary approaches address different considerations and constraints: hybrid schemes balance accuracy with transmission costs through lossy/lossless models; lossless time-series compression exploits the natural temporal relation in continuous sensor readings; and two-tier architectures apply different techniques at both the sensor and gateway level to reduce energy costs.

Hybrid Lossy/Lossless Compression

The hybrid compression scheme presented in [12] addresses accuracy requirements by separating real-time lossy transmission from on-demand lossless reconstruction. The Fan

algorithm adaptively sub-samples data maintaining bounded error ε, achieving average 7.8× and 2.1× Compression ratio (CR) for lossy and lossless compression, respectively.

The approach provides graceful degradation through battery-aware switching, in that when battery drops below a certain threshold (e.g., 20%), the system switches to lossy-only transmission instead of lossless, extending lifetime at the cost of reconstruction accuracy.

This trade-off aligns with disaster-scenario requirements where sustained operation during prolonged periods of network outages often takes higher priority over perfect data-fidelity, as approximate sensor readings over extended periods of time provide greater situational awareness than high-precision measurements from devices that quickly deplete their batteries. However, safety-critical sensors (e.g., structural integrity sensors, hazardous gas detectors) may require different thresholds (if any) or other policies to balance longevity with accuracy requirements.

Lossless Time-Series Compression

Sprintz [7] targets lossless compression for multivariate integer time-series with extreme resource constraints, achieving 2-10× compression ratios with <1KB memory footprint and 8-sample block sizes. The four-component algorithm combines forecasting (delta coding or Fast Integer REgression (FIRE) online learning), bit-packing with zigzag encoding, Run-length encoding (RLE) for all-zero blocks, and optional Huffman coding.

Delta coding is extremely fast, and when combined with RLE becomes even more so, as it yields a run of zero errors if the data is constant, which is likely to happen for nominal sensor readings. FIRE forecasting yields better compression, but its online learning approach place it beyond the scope of this paper.

A *forescaster* is employed to predict each sample based on previous ones, and the difference between the next and the predicted sample are encoded. This difference is typically closer to zero than the next sample itself.

Any prediction errors from the previous forecasting step are zigzag encoded as a "payload", and a header is prepended with sufficient information to invert the bit packing.

If a block happens to consist only of zeros, this approach waits for a block in which there is a non-zero error, and the number of all-zero blocks are written out using RLE instead of the (empty) payload.

Finally, the bit packed representation of each block can be *entropy coded* using a Huffman coder, applied to the headers and payloads. This is done after bit packing because not only is it faster, but it also increases compression.

Decompression can achieve up to 3GB/s throughput without Huffman coding (>500MB/s with Huffman) in a single thread, enabling high-speed data retrieval on gateway-tier devices. Compression maintains >200MB/s on 8-bit data, sufficient for real-time operation even on embedded platforms.

Two-Tier Compression Architecture

The two-tier data reduction technique proposed in [30] applies compression at both sensor nodes (Tier 1) and gateways (Tier 2) to reduce energy consumption in the overall system.

Sensor nodes employ Delta encoding followed by RLE, allowing this approach to achieve 80-84% compression by exploiting the temporal correlation in sensor data. Gateways perform hierarchical clustering based on the Minimum description length (MDL) principle – that is, combining pairs of datasets into one cluster when their combined representation is shorter than separate encodings – to transmit *hypothesis* data sets and difference vectors instead of full data sets.

FIXED: descrever o MDL um bocado

The Delta+RLE implementation – validated through OMNeT++ simulation – proves to be lightweight enough for resource-constrained nodes, while gateway clustering reduces transmission to cloud infrastructure.

2.5.3 Discussion

Existing data reduction techniques demonstrate substantial energy savings on resource-constrained platforms but assume single-protocol deployments.

Ambrosia's [38] window-based prediction achieves 60% data reduction with 99% faster execution than ARIMA while maintaining accuracy within application-specific error thresholds, validated on 8-bit microcontrollers.

Sprintz [7] provides lossless time-series compression with 2-10× compression ratios using <1KB memory and 8-sample blocks, combining delta coding, zigzag encoding, RLE, and optional Huffman coding with >200MB/s compression throughput suitable for real-time embedded operation.

Hybrid lossy/lossless [12] schemes enable graceful battery-aware degradation, while two-tier architectures [30] (Delta+RLE at sensors, MDL clustering at gateways) achieve 80-84% compression exploiting temporal correlation.

None of these approaches address protocol heterogeneity by themselves, and different wireless technologies have distinct energy costs and range/accuracy trade-offs that warrant protocol-specific compression/reduction strategies.

We have found no work that integrates prediction-based reduction with protocol-aware adaptive thresholding and compression tuning for heterogeneous multi-protocol networks.

μ-Babel's approach (Section 3.5.4) should, therefore, consider lightweight prediction and compression in multi-protocol scenarios with per-protocol thresholds and strategy selection.

2.6 Summary

Table 2.2 summarizes the key architectural differences between existing work and μ-**Babel**'s planned vision across the technical domains discussed in this chapter.

Table 2.2: Related work summary

Domain	**Related Work Limitations**	μ-**Babel** Approach
Protocol coordination	Master nodes (MCSC-WoT), SDN controllers (MINOS), gateway coordinators (multi-protocol gateway)	Fully distributed selection
Mesh topology	Tree-based restrictions, TCP-dependent links (HyParView), centralized coordination	Connectionless BLE/LoRa discovery, hybrid views, true P2P routing
Time sync	Master beacons (MCSC-WoT), single-protocol gossip (RGCS)	Protocol-specific Poisson rates, multi-graph converge-to-max
Compression	Single-protocol optimization (Ambrosia, Sprintz), sender-focused	Protocol-aware adaptive thresholds, gateway receiver energy management

FIXED: removed seurity row

A common pattern is present across these domains: existing approaches rely on centralized coordination mechanisms that become single points of failure during infrastructure collapse.

Protocol coordination depends on master nodes or SDN controllers, mesh topologies assume consistent connectivity or gateway coordinators, time synchronization requires master beacons, and authentication protocols depend on centralized servers or group leaders.

Even distributed approaches like HyParView and RGCS make assumptions (routing infrastructure, persistent TCP connections, single-protocol homogeneity) incompatible with resource-constrained multi-protocol disaster scenarios.

In the next Chapter we discuss how these limitations inform the design of μ-**Babel** as motivating factors for developing a fully decentralized architecture – one that prioritizes autonomous operation, protocol diversity, and resilience under unstable connectivity.

FIXED: closing paragraph, mais narrativa etc?

There we highlight the system requirements, operating conditions, and architectural choices that will enable robust communication and coordination across heterogeneous devices.

3

Solution Architecture

removed recap for space.....

3.1 System Requirements

3.1.1 Overview of μ-Babel's Approach

In order to effectively overcome the aforementioned challenges, we can highlight a set of Functional Requirements (FRs) that **μ-Babel** must meet in order to eliminate the *dependency* on centralized coordination, continuous connectivity, and cloud infrastructure; and a set of Non-Functional Requirements (NFRs) which would strengthen the overall quality of the framework and ensure it can be widely adopted with ease.

Functional Requirements

"something something these are essential ..." ou o parágrafo anterior basta?

Infrastructure Independence: The system must maintain communication capabilities when traditional infrastructure becomes unavailable. Rather than treating this unavailability as an exceptional condition requiring failover mechanisms, our architecture will focus on autonomous Peer-to-peer (P2P) connectivity as the baseline mode of operation, with infrastructure integration as an added benefit when available, rather than a hard dependency.

Autonomous Operation: Devices will self-organize into functional networks, dispensing centralized coordinators and static master nodes. Instead, local P2P gossip interactions will be the basis for coordination and synchronization.

Multi-Protocol Exploitation: Communication strategies will adapt to the available mediums and device resources without centralized control. Communications will continue despite disruption of individual protocols through adaptive selection and opportunistic protocol switching. Protocol diversity provides resilience through redundancy, and this should be achieved with minimal impact on the upper layers of the application (through abstractions).

Scalability: The system must support deployments ranging from small, localized networks to large-scale, highly distributed collections. Rather than requiring global knowledge of all participants, nodes organize into local groups where elected leaders maintain state only for their immediate members. This area-based approach – analogous to Open Shortest Path First (OSPF)'s autonomous systems [27] – distributes coordination load and enables networks to scale while keeping per-node overhead bounded by local group size.

Resource Efficiency: Embedded platforms inevitably present strict resource constraints (i.e., memory storage, processing, battery), thus all protocols and algorithms must execute within these limitations. In order to enable this, lightweight data compression and reduction techniques will be employed to make the most out of the limited storage and battery available, and adaptive protocol selection will take into account device capabilities and available resources.

Non-Functional Requirements

"something something these are not essential ..." ou o parágrafo inicial basta?

Optional Cloud Integration: When infrastructure *is* available, it should be taken advantage of. Thus, **μ-Babel** will exploit it for tasks that are beyond the capabilities of individual nodes or even gateways, such as data aggregation, analytics, and long-term storage. Cloud integration thus becomes an optional – and welcome – addition to the repertoire of features, but does not itself become a requirement for dependable operation.

Interoperability: Following optional cloud integration, widely used Internet of Things (IoT) protocols such as MQTT and Constrained Application Protocol (CoAP) can be leveraged to enable integration with existing IoT platforms and services when conditions allow.

3.2 Operating Conditions and Failure Modes

μ-Babel is designed to remain operational under conditions where traditional infrastructure becomes unreliable or unavailable. This section establishes the operational environment and failure scenarios that drive our architectural decisions.

3.2.1 Target Operating Environments

move this para os use cases later?

μ-Babel targets deployments where infrastructure availability cannot be guaranteed, and in which autonomous operation is a necessity:

Disaster Scenarios: Locations susceptible to natural disasters (i.e., earthquakes, floods, fires) or infrastructure failures that eliminate power grids and network access points.

Remote or Hazardous Locations: Industrial sites, wilderness monitoring, or dangerous environments where manual intervention and infrastructure maintenance are impractical or impossible.

Local-First Deployments: Environments where data sovereignty requirements or privacy concerns require local processing and control rather than cloud dependency, even when connectivity is available.

3.2.2 Expected Failure Modes

The architecture must maintain functionality under the following conditions:

Infrastructure Collapse: Complete loss of access points and external network connectivity. Devices must self-organize and maintain P2P communication without infrastructure support.

Communication Medium Disruption: Individual protocols may become unavailable due to interference, jamming, congestion, or physical obstacles blocking signal propagation. Multi-protocol diversity provides alternative communication paths and ensures continued operation.

Network Partitioning: Physical damage or communication range limitations may split networks into isolated subgroups. Devices within partitions must continue autonomous operation while supporting eventual reconciliation when connectivity is restored.

Node Failures: Individual devices may become unavailable due to battery depletion, physical damage, or environmental conditions. The network must maintain operation despite individual node failures without depending on specific coordinator nodes.

Intermittent Connectivity: Communication links may be unstable with varying packet loss, latency, and availability. Protocols must tolerate unreliable links and adapt to changing conditions.

3.3 Hardware Platform

μ-Babel targets resource-constrained embedded platforms commonly used in IoT and domotics systems. The following sections cover device specifications and communication protocol characteristics, establishing the foundation for our multi-protocol, heterogeneous architecture.

We distinguish between two device classes based on their capabilities and roles within the architecture.

3.3.1 Class-1: Battery-Powered Sensor Nodes

The primary targets for **μ-Babel**. These embedded devices handle sensing, actuation, P2P communication, and decentralized coordination without requiring external infrastructure.

Table 3.1 presents the resources available across Class-1 devices. These span from basic sensor nodes (Pico 2W, ESP32-C5/C6) to more capable embedded devices (ESP32-S3/P4), enabling role differentiation within the autonomous peer network.

TODO: figure out table spacing

Table 3.1: Class-1 Device Specifications

Device	CPU[1]	CPU Clock	SRAM	PSRAM[2]	Flash
Raspberry Pi Pico 2 W	Dual-Core Arm Cortex-M33	150 MHz	520 KB	✗	4 MB
ESP32	Dual-Core Xtensa LX6	240 MHz	520 KB	✗	16 MB
ESP32-C5	Single-Core RISC-V	240 MHz	384 KB	✗	4 MB
ESP32-C6	Single-Core RISC-V	160 MHz	512 KB	✗	4 MB
ESP32-S3	Dual-Core Xtensa LX7	240 MHz	512 KB	8 MB	16 MB
ESP32-P4	Dual-Core RISC-V	360 MHz	768 KB	32 MB	32 MB

[1] All supported devices are 32-bit
[2] Pseudostatic random-access memory (PSRAM) is supported on all devices, but not available in the particular models used during development

Resource availability within Class-1 devices varies significantly both in processing power and available memory. While there are no fixed roles, capability awareness naturally influences dynamic selection procedures:

Data Aggregation and Buffering: Higher-resource devices (ESP32-S3/P4) can maintain larger message buffers and aggregate data from multiple sensor nodes, which is particularly valuable during network partitions when store-and-forward mechanisms come into effect.

Protocol Bridging: Devices with broader protocol support naturally serve as bridges between protocol domains, forwarding messages across different mediums based on advertised capabilities.

Protocol Diversity and Device Coverage

Table 3.2 shows the communication protocols supported across Class-1 devices, demonstrating considerable heterogeneity.

Protocol support across devices provides several advantages:

Coverage: All devices support at least three protocols (Wi-Fi, Bluetooth Low Energy (BLE), LoRa), ensuring baseline multi-protocol operation across the entire platform range. Even the least capable device (Pico 2 W) can participate in protocol-diverse mesh networks.

Table 3.2: Supported Communication Protocols

Device	Wi-Fi	Bluetooth	ESP-NOW	ZigBee	LoRa[1]
Pico 2W	2.4GHz	Classic + Low Energy (LE) 5.2	✗	✗	✓
ESP32	2.4 GHz	Classic + LE 4.2	✓	✗	✓
ESP32-C5[2]	2.4/5 GHz	LE 6.0	✓	3.0	✓
ESP32-C6[2]	2.4 GHz	LE 5.3	✓	3.0	✓
ESP32-S3	2.4 GHz	LE 5.0	✓	✗	✓
ESP32-P4[3]	2.4 GHz	LE 5.3	✓	✗	✓

[1] Via external SX1276/SX1262 transceiver, controlled through Serial Peripheral Interface (SPI)
[2] ESP32-C5/C6 also support 802.11ax (Wi-Fi 6) and Thread, but we focus on ZigBee given its wider adoption in IoT systems
[3] ESP32-P4 does not possess wireless capabilities by itself and relies on a C6-Mini coprocessor, with ZigBee support still in development

Protocol-Specific Advantages: ESP32 family devices can leverage ESP-NOW for P2P communication without infrastructure. ESP32-C5/C6 additionally provide ZigBee support for standardized mesh networking – particularly interesting in building automation and domotics contexts.

Notably, ESP32-C5 supports ESP-NOW over both 2.4 and 5 GHz bands, which provides a crucial short-range alternative path for congested environments where BLE, Wi-Fi and ZigBee all contend for 2.4 GHz channels.

Long-Range Capabilities: All platforms support external LoRa transceivers via SPI, enabling long-range communication for disaster scenarios. This addresses the multi-protocol resilience requirements discussed in Section 2.3.

These factors combine to provide multi-protocol resilience: spectrum diversity across 2.4/5 GHz and sub-GHz bands, infrastructure-free operation via ESP-NOW and BLE, and long-range connectivity through LoRa.

Together, these Class-1 specifications and multi-protocol capabilities establish the core hardware foundation for **μ-Babel**'s autonomous operation. Section 3.5 details how these resources are exploited in the proposed architecture.

3.3.2 Class-2: Optional Aggregation Nodes

Resource-rich devices that run the broader Babel [16] framework and are mentioned here for the sake of integration with broader systems, but are not required for **μ-Babel**'s core autonomous operation.

This class encompasses general-purpose computing devices such as desktops, laptops, and servers. For development purposes, and staying within the realm of embedded platforms, we specifically utilize Raspberry Pi Single-board computers (SBCs), whose specifications are available in Table 3.3.

Class-2 devices serve as *optional* infrastructure that, when present, can perform: (1) Data aggregation and long-term storage; (2) Computationally intensive analytics beyond

Table 3.3: Class-2 Device Specifications

Device	CPU[1]	CPU Clock	RAM[1]	Storage
Raspberry Pi 5	Quad-Core 64-bit Arm Cortex-A76	2.4 GHz	4-16 GB	Variable external
Raspberry Pi 4	Quad-Core 64-bit Arm Cortex-A72	1.8 GHz	4-8 GB	Variable external

[1] Configurations with less Random-access memory (RAM) are available but were not considered

Class-1 capabilities; and (3) Optional cloud bridging when external connectivity is available and desired (e.g., MQTT brokerage, CoAP endpoints).

Critically, these devices *do not* serve as coordinators or single points of failure. **μ-Babel** networks will operate fully autonomously with Class-1 devices alone; Class-2 nodes simply participate as peers that happen to have greater resources. Their absence does not prevent network formation, peer discovery, or autonomous operation.

3.4 Software Stack and Development Environment

μ-Babel is implemented on top of well-established embedded software platforms that provide hardware abstraction and concurrency support while remaining sufficiently lightweight for resource-constrained platforms.

The system leverages vendor-supported Software Development Kits (SDKs) and a minimal Real-time operating system (RTOS) to ensure portability across Class-1 devices while maintaining fine-grain control over hardware resources.

3.4.1 Operating System Layer

FreeRTOS [4] serves as the RTOS across all Class-1 devices. It provides deterministic task scheduling and inter-task communication implemented around a single queue primitive [5]. This queue mechanism forms the basis for queues (as data structures), semaphores, and mutexes. FreeRTOS also allows for configurable memory allocation and per-task stack management, all without the overhead of a full-featured Operating system (OS).

This allows multiple protocol components and I/O handlers to execute concurrently while preserving predictable timing behavior, which is essential for networking and radio-driver applications.

FreeRTOS is the *de facto* OS on ESP32-family devices, and is easily brought into the Pico platform. By using it as a common execution runtime, **μ-Babel** maintains a consistent concurrency and timing model across heterogeneous microcontroller platforms, simplifying both portability and implementation.

3.4.2 Platform SDKs and Hardware Abstractions

Hardware-specific functionality is accessed through vendor abstractions. The Espressif IoT Development Framework (ESP-IDF) [14] is used for the ESP32 family, while the Raspberry

Pi Pico SDK [31] is used for RP2350-based devices.

These frameworks provide Application programming interfaces (APIs) for peripheral access (e.g., General-purpose input/output (GPIO), SPI, Inter-Integrated Circuit (I^2C), Universal asynchronous receiver-transmitter (UART)), timers, interrupt handling, Wi-Fi/BLE radio stacks, and – where available – hardware-accelerated cryptographic operations.

By encapsulating platform-specific bare-metal details behind stable interfaces, they allow **μ-Babel** to target multiple microcontrollers through a set of thin Hardware Abstraction Layers (HALs), enabling re-utilization of higher-level protocols and coordination logic across Class-1 devices. **μ-Babel** is implemented in C, as it allows precise control over memory usage, interrupt behavior, and timing, which are essential in embedded environments with limited memory strict real-time constraints.

3.4.3 Concurrency Model

μ-Babel follows a task-based concurrency model built on FreeRTOS primitives. Protocol handlers execute as independent tasks, allowing computation, I/O, and radio operations to execute concurrently and predictably. Inter-task communication is handled through message queues, while mutexes and semaphores are used to protect shared state and coordinate access to hardware resources.

An event dispatcher provides higher-level coordination by routing events between protocol components and system services. This structure decouples individual components, simplifies reasoning about system behavior, and supports incremental extension of the protocol stack.

3.5 System Model

μ-Babel is structured as a layered architecture where components interact to achieve autonomous multi-protocol operation. This section presents the key architectural components organized into four functional layers: Core Components, Discovery & Topology, Coordination, and Data Management.

figure out onde meter o event dispatcher?

Figure 3.1 illustrates the overall component organization and interactions.

3.5.1 Core Components Layer

mix entre o que temos e o que podemos vir a ter

The Core Components Layer provides the foundational services for protocol management, event dispatching, and connection handling. These components abstract protocol heterogeneity and provide unified interfaces to higher layers.

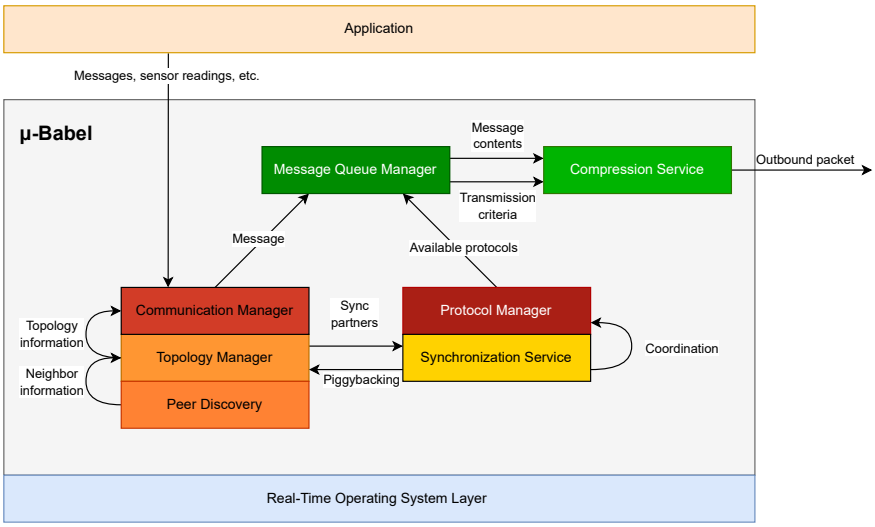

Figure 3.1: System architecture and component interactions

Protocol Manager

The Protocol Manager maintains runtime information for all available communication protocols and device capabilities. It registers available protocols during system initialization and tracks their operational status throughout device operation, simultaneously making decisions on adaptive selection based on current operational conditions.

Key responsibilities include: (1) Registering protocol handlers and their characteristics (range, throughput, energy cost); (2) Tracking which protocols are supported on each device based on hardware capabilities; (3) Enabling or disabling protocols dynamically based on runtime conditions (e.g., battery level, interference detection); (4) Selecting the appropriate protocol(s) for messages based on Quality of Service (QoS), device capabilities, energy budget and environmental conditions, all while preventing excessive oscillation through switching cost awareness

Event Dispatcher

The Event Dispatcher implements an event-driven coordination model that decouples components and enables asynchronous communication between services. Components register interest in specific event types and subtypes, and the dispatcher routes incoming events to all registered subscribers.

This approach prevents tight coupling between protocol implementations and application logic, enabling independent development and testing of individual components while maintaining system-wide coordination.

Communication Manager

The Communication Manager provides protocol-agnostic virtual connection state tracking and high-level abstractions for peer communication. Applications and system components interact with remote nodes using Universally unique identifiers (UUIDs) rather than protocol-specific addresses.

For each virtual connection, the manager: (1) Maintains UUID-to-protocol mappings; (2) Monitors per-protocol link availability; (3) Determines message routing by querying the Topology Manager; (4) Hands messages to the Message Queue Manager with routing decisions.

This unified abstraction enables higher-layer components to interact with peers without concerning themselves with protocol-specific details.

3.5.2 Discovery & Topology Layer

wishful thinking

The Discovery & Topology Layer handles peer discovery, network topology maintenance, and multi-hop routing. These components enable autonomous network formation without centralized coordination.

Peer Discovery Service

The Peer Discovery Service implements multi-protocol neighbor discovery. Short-range protocols like BLE, ESP-NOW, and ZigBee employ active discovery mechanisms (i.e., advertising, peer lists, beacons), while long-range LoRa communication relies on opportunistic message exchange.

The service discovers peers and provides standardized information to the Topology Manager by: (1) Periodically scanning for neighbor advertisements and beacons across active protocols; (2) Advertising local device capabilities (supported protocols, computational resources, battery status); (3) Translating protocol-specific discovery information into standardized peer descriptors for the Topology Manager.

Topology Manager

Inspired by HyParView [21], the Topology Manager maintains hybrid partial views for scalability and resilience. Each node keeps a small *active view* of peers with which it maintains active communication links, and a larger *passive view* serving as a pool of potential neighbors.

The active view is managed reactively: nodes are added during join operations and removed upon failure detection. The passive view is maintained through periodic shuffle operations that exchange peer descriptors with neighbors, providing view diversity without global knowledge.

Key operations include: (1) Managing peer membership in active and passive views; (2) Passive view shuffling for backup peer discovery; (3) Providing routing decisions to

Communication Manager; (4) Automatic peer promotion from passive to active view upon failures;

3.5.3 Coordination Layer

wishful thinking too

The Coordination Layer manages time synchronization, enabling coordinated operation without master nodes.

Synchronization Service

Coordinated multi-protocol operation requires time synchronization for channel hopping and scheduled communication windows. The Synchronization Service implements decentralized consensus on clock values through randomized gossip, inspired by Randomized gossip-consensus-based sync (RGCS) [43].

Each node maintains a logical clock composed of rate and offset parameters that transform its hardware clock into synchronized logical time. Synchronization occurs through gossip exchanges using Poisson-distributed intervals to minimize collision probability, with sync partners selected from the Topology Manager's active view. Sync messages can also be piggybacked on the Topology Manager's shuffle exchanges to reduce communication overhead.

explicar melhor what I mean by Poisson-distributed?

Key aspects include: (1) Per-protocol Poisson gossip rates accounting for protocol-specific energy costs and reliability; (2) Converge-to-max criterion for faster convergence than average-based approaches; (3) Simultaneous rate and offset compensation; (4) Multi-protocol operation where sync messages can be sent over any available protocol.

Unlike master-based synchronization that creates single points of failure, this gossip-based approach maintains synchronization through distributed local interactions.

3.5.4 Data Management Layer

wishful thinking too[2]

The Data Management Layer optimizes data transmission through compression and efficient queue management, extending operational lifetime in battery-powered deployments.

Compression Service

The Compression Service reduces transmission overhead through prediction-based data reduction (inspired by Ambrosia [38]) and lightweight compression (inspired by Sprintz [7] and the approach in [30]). It employs window-based forecasting where sensor values are transmitted only when prediction error exceeds application-specific thresholds.

Both sender and receiver maintain synchronized prediction state, ensuring consistency without additional communication overhead. The service supports: (1) Prediction-based reduction with configurable error thresholds (δ); (2) Protocol-aware threshold adaptation

(higher thresholds for energy-expensive protocols); (3) Lightweight time-series compression using delta coding, zigzag encoding, and Run-length encoding (RLE); (4) Graceful degradation under low battery conditions.

Protocol-specific compression strategies account for varying energy costs: data transmitted over LoRa may employ more aggressive compression than BLE transmissions, balancing compression overhead against transmission savings.

Message Queue Manager

The Message Queue Manager handles all message transmission scheduling. It receives routing decisions from the Communication Manager and coordinates with the Protocol Manager to determine when messages can actually be transmitted. It maintains separate queues for different message priorities (e.g., critical alerts, periodic sensor data, opportunistic synchronization) and enforces memory limits based on device capabilities.

Responsibilities include: (1) Priority-based queuing with configurable policies; (2) Store-and-forward buffer management for partitioned networks; (3) Per-protocol transmission scheduling based on protocol availability; (4) Delivery confirmation tracking and retransmission scheduling; (5) Message aging awareness;.

3.5.5 Component Interactions

Peer Discovery continuously feeds neighbor information to the Topology Manager, which maintains active/passive views and provides topology information to the Communication Manager for routing decisions. The Communication Manager then hands messages to the Message Queue Manager for immediate transmission or buffering based on those decisions.

The Message Queue Manager coordinates with the Protocol Manager to schedule actual transmission by querying for protocol availability and managing the transmission timing, the Compression Service then acts upon outbound messages prior to actual transmission, performing data reduction if the message type and transmission method allow for or require it.

The Synchronization Service coordinates with the Protocol Manager to prevent protocol switching during synchronization exchanges, and selects sync partners from the Topology Manager's active view. Additionally, it can opportunistically coordinate during shuffle events by piggybacking synchronization information on peer description exchanges, thus reducing the total gossip overhead.

The Event Dispatcher orchestrates asynchronous inter-component communication, enabling loosely coupled interactions through event publication and subscription, simplifying component development and incremental system evolution.

4

Progress Report and Planning

This chapter outlines the current status of the proposed implementation, the validation strategy, and development timeline for **μ-Babel**.

4.1 Progress Report

The implementation of **μ-Babel** is currently in its early stages, with protocol-specific handlers under development.

The Core Infrastructure Layer (Section 3.5.1) is partially complete. Hardware Abstraction Layers (HALs) for peripheral access (General-purpose input/output (GPIO), Serial Peripheral Interface (SPI) and Inter-Integrated Circuit (I^2C)) are functional across both ESP-IDF and Pico SDK platforms; the Event Dispatcher is operational, providing event-driven communication between components, and the Communication Manager currently supports Wi-Fi peer connection management, and unicast/multicast message delivery.

The Discovery & Topology Layer (Section 3.5.2) has one working component in the Peer Discovery Service, which currently supports discovery over Wi-Fi.

The Coordination (Section 3.5.3), and Data Management (Section 3.5.4) Layers remain in the planning and design phase.

Proof-of-Concept Deployment: The current system has been deployed and tested on M5Stack Core Basic [23] devices (ESP32-based) by commanding multiple Raspberry Pi 5 boards attached to actuators, demonstrating an "inversion of control" to signal the flexibility and potential of the proposed system.

4.1.1 Use Cases

To guide development and validate **μ-Babel**'s implementation, we consider two complementary deployment scenarios that showcase different aspects of the system's capabilities. These use cases are meant to demonstrate operation during normal and simulated disaster conditions, taking advantage of a heterogeneous collection of devices (i.e., resource-constrained sensors and more capable gateways) with optional cloud integration).

Campus-Wide Environmental Monitoring

Deployment Context: A university campus with sensors distributed across multiple buildings to monitor environmental conditions (e.g., temperature, humidity, CO_2 levels, occupancy) and classroom occupancy. The deployment consists of:

TODO: look into Millimeter Wave Radar

- **Class-1 sensor nodes:** Battery-powered devices (ESP32-family, Pico 2 W) placed throughout rooms and corridors, performing periodic sensing
- **Class-1 gateway nodes:** Battery-powered devices with greater resources (ESP32-S3/P4) at strategic locations within each building, serving as protocol bridges and data aggregators
- **Class-2 aggregation nodes:** Grid-powered Raspberry Pi devices in server rooms or management offices, providing additional processing capacity and optional cloud connectivity

Normal Operation Mode: During stable conditions, the system operates with cloud connectivity when available, with Class-2 devices acting as MQTT brokers and makeshift cloud servers. Sensor nodes form local meshes within buildings, forwarding data through gateway nodes to building-level aggregators.

This mode validates: (1) Multi-protocol communication for different ranges: Bluetooth Low Energy (BLE) within rooms, ESP-NOW/ZigBee between floors, LoRa across buildings; (2) Topology management with hybrid active/passive views; (3) Data reduction for efficient transmission; (4) Optional cloud integration (Non-Functional Requirement (NFR), Section 3.1.1).

Disaster Operation Mode: Upon infrastructure failure (simulated power outage or network collapse), battery-powered sensors continue autonomous operation. The system maintains local coordination through Peer-to-peer (P2P) gossip interactions, with Class-1 gateways aggregating data from their vicinity and coordinating between buildings via LoRa.

This mode validates: (1) Infrastructure independence (Functional Requirement (FR), Section 3.1.1); (2) Autonomous operation without cloud connectivity; (3) Protocol adaptation when primary mediums become unavailable; (4) Store-and-forward mechanisms; (5) Resource-efficient operation on limited battery power.

Building Safety and Evacuation System

Deployment Context: A safety-focused monitoring system, combining environmental hazard detection (smoke, fire, CO_2, structural integrity) with occupancy tracking and evacuation guidance. This use case expands upon the campus scenario by emphasizing real-time responsiveness and critical message prioritization:

a "earthquake detection" de SCMU não funcionava muito bem... maybe outra coisa?

- **Class-1 hazard sensors:** Fixed sensors at key locations (stairwells, corridors, rooms) for smoke, temperature, and air quality monitoring
- **Class-1 gateway nodes:** Per-floor devices maintaining floor-level mesh coordination and hazard aggregation
- **Class-2 building coordinator:** Central aggregation point integrating with existing building management systems when available
- **Mobile display nodes:** Battery-powered displays (ESP32-S3/P4 with e-paper or small screens) showing evacuation routes and hazard warnings

Normal Operation Mode: The system continuously monitors environmental conditions and occupancy patterns, integrating with building management infrastructure for logging and analysis. Sensor readings are transmitted via short-range protocols (BLE, ESP-NOW) to floor gateways, which forward aggregated data to the building coordinator.

This mode validates: (1) Priority-based message queuing (critical vs. periodic data); (2) Low-latency local communication for real-time monitoring; (3) Integration with existing infrastructure.

Disaster Operation Mode: Upon detecting hazardous conditions or infrastructure failure, the system automatically transitions to emergency mode. Hazard sensors immediately broadcast critical alerts via available protocols, floor gateways compute safe evacuation routes based on detected hazards, and displays show real-time evacuation guidance reflecting current conditions.

might be over-promising, cada um teria de manter some sort of layout info...

ou.... layout info stored in an SD card? much to consider

This mode validates: (1) Autonomous operation during infrastructure collapse; (2) Multi-protocol broadcast for critical alerts (redundancy through protocol diversity); (3) Decentralized coordination without central controllers; (4) Adaptive protocol selection under emergency conditions; (5) Real-time responsiveness despite limited resources.

4.2 Planning

4.2.1 Validation

To validate μ-**Babel**'s implementation, we plan to follow a three-tier approach: component verification, integration testing, and deployment evaluation.

Component Verification: Individual components are validated in isolation to ensure correct behavior under controlled conditions:

- **Protocol handlers:** Message delivery rates, latency bounds, and energy consumption per protocol

- **Peer discovery:** Discovery time and neighbor view consistency across protocol combinations
- **Topology manager:** Active/passive view maintenance and failure detection latency
- **Synchronization service:** Clock drift compensation, gossip convergence speed, and multi-protocol sync overhead
- **Compression service:** Compression ratios, prediction accuracy, and computational overhead on Class-1 devices

algo sobre validar energy costs?

Integration Testing: System properties are evaluated for different metrics:

- **Infrastructure independence:** Network formation time without access points or coordinators
- **Protocol adaptation:** Switching latency and stability under changing conditions (e.g., interference, protocol unavailability)
- **Fault tolerance:** Recovery time after node failures, message delivery during network partitions
- **Scalability:** Performance degradation as network size increases
- **End-to-end latency:** Message delivery times across multi-hop paths with varying protocol combinations

Deployment Evaluation: Real-world validation using the use cases from Section 4.1.1:

- **Campus monitoring scenario:** Multi-building deployment measuring autonomous operation duration, data delivery rates, and battery lifetime under normal vs. disaster modes
- **Building safety scenario:** Single-building deployment measuring critical alert propagation, evacuation guidance system, and responsiveness during simulated infrastructure failures

Where applicable, performance will be compared against centralized alternatives (e.g., MQTT based systems) to quantify the benefits and drawbacks of our approach.

4.2.2 Task Scheduling

In order to achieve the proposed requirements (Section 3.1.1), we delineate a set of tasks to be carried out in the coming months:

Task 1 – Core Components: Foundational protocol and communication primitives

- **Task 1.1:** Protocol Manager
- **Task 1.2:** Event Dispatcher Validation
- **Task 1.3:** Communication Manager

Task 2 – Discovery & Topology: Network formation, discovery and routing mechanisms

- **Task 2.1:** Peer Discovery Service
- **Task 2.2:** Topology Manager

Task 3 – Coordination Layer: Coordination and Synchronization mechanisms

- **Task 3.1:** Synchronization Service

Task 4 – Data Management : Lightweight reduction and compression strategies

- **Task 4.1:** Message Queue Manager
- **Task 4.2:** Compression Service

Task 5 – Integration

Task 6 – Validation: Following the approach outlined in Section 4.2.1

- **Task 6.1:** Use Case Development
- **Task 6.2:** Experimental Setup
- **Task 6.3:** Performance Evaluation

I mean duh... se calhar retiro isto, era a pensar no integration hell ser chato e demorado de resolver

Task 7 – Dissertation Work: Elaboration of the final document

- **Task 7.1:** Writing
- **Task 7.2:** Preparation for PerCom (2027)

perhaps too ambitious...?

The GANTT chart in Figure 4.1 presents the timeline for completing these tasks.

needs to be updated

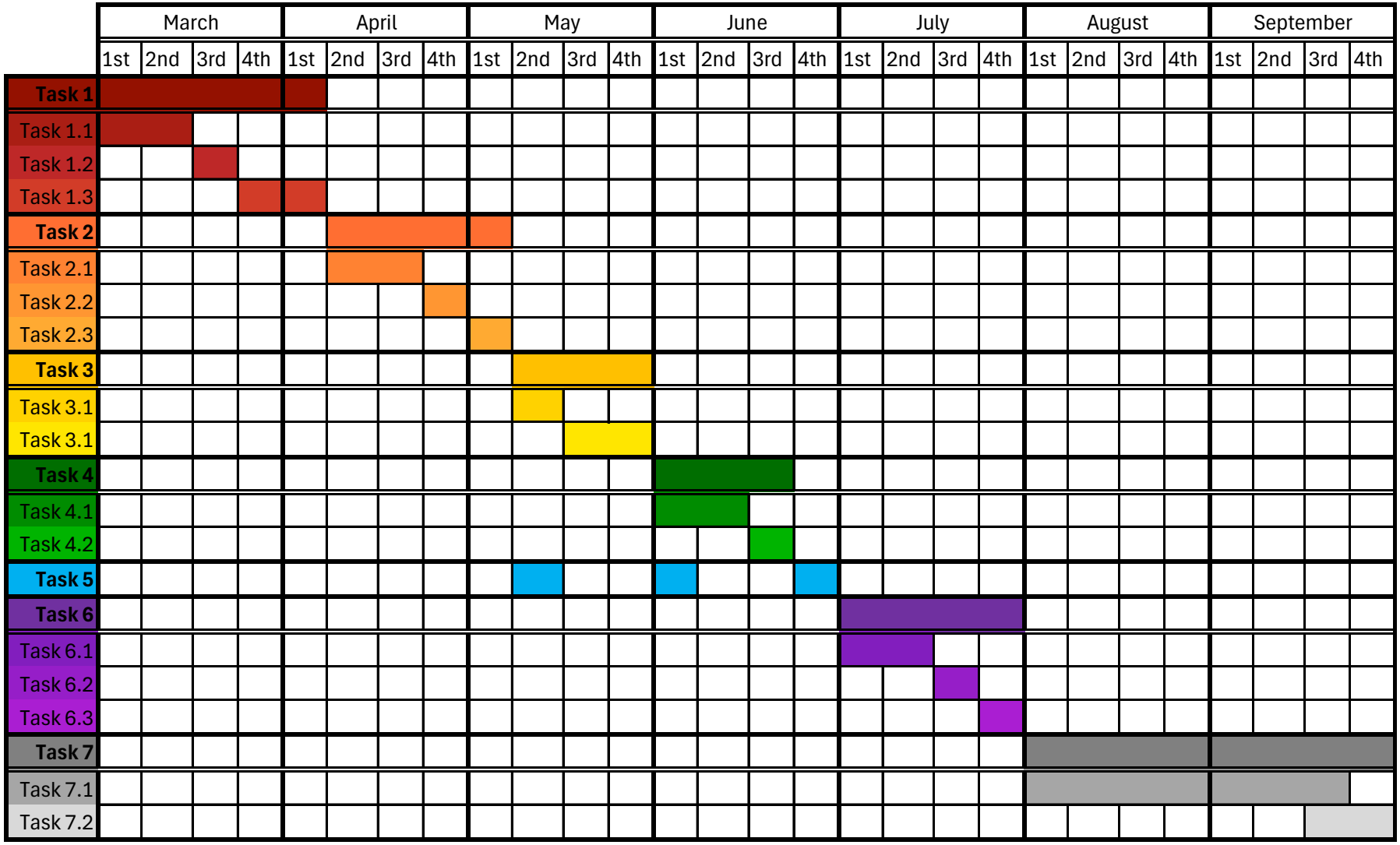

Figure 4.1: Proposed timeline

Bibliography

[1] S. M. Alamouti, F. Arjomandi, and M. Burger. "Hybrid Edge Cloud: A Pragmatic Approach for Decentralized Cloud Computing". In: *IEEE Communications Magazine* 60.9 (2022), pp. 16–29. DOI: 10.1109/MCOM.001.2200251 (cit. on p. 1).

[2] R. Alexander et al. *RPL: IPv6 Routing Protocol for Low-Power and Lossy Networks*. RFC 6550. 2012-03. DOI: 10.17487/RFC6550 (cit. on p. 14).

[3] Z. Amiri et al. "Resilient and dependability management in distributed environments: A systematic and comprehensive literature review". In: *Cluster Computing* 26.2 (2023), pp. 1565–1600 (cit. on p. 2).

[4] AWS open source. *FreeRTOS*. 2024. URL: https://www.freertos.org/ (visited on 2026-01-27) (cit. on p. 29).

[5] AWS open source. *Intertask Communications*. 2026. URL: https://www.freertos.org/message_passing_performance (visited on 2026-01-27) (cit. on p. 29).

[6] P. Barman, R. Chowdhury, and B. Saha. "Multi-channel secure communication framework for wireless IoT (MCSC-WoT): enhancing security in Internet of Things". In: *Cluster Computing* 28.11 (2025), p. 691 (cit. on pp. 15–18).

[7] D. Blalock, S. Madden, and J. Guttag. "Sprintz: Time Series Compression for the Internet of Things". In: *Proc. ACM Interact. Mob. Wearable Ubiquitous Technol.* 2.3 (2018-09). DOI: 10.1145/3264903 (cit. on pp. 21, 22, 33).

[8] C. Bormann, A. P. Castellani, and Z. Shelby. "CoAP: An Application Protocol for Billions of Tiny Internet Nodes". In: *IEEE Internet Computing* 16.2 (2012), pp. 62–67. DOI: 10.1109/MIC.2012.29 (cit. on p. 8).

[9] R. Casadei et al. "Engineering Resilient Collaborative Edge-Enabled IoT". In: *2019 IEEE International Conference on Services Computing (SCC)*. 2019, pp. 36–45. DOI: 10.1109/SCC.2019.00019 (cit. on pp. 12, 13).

[10] M. Chui, M. Collins, and M. Patel. *The Internet of Things: Catching up to an accelerating opportunity*. McKinsey & Company. 2021-11. URL: https://www.mckinsey.com/capabilities/mckinsey-digital/our-insights/iot-value-set-to-accelerate-through-2030-where-and-how-to-capture-it (visited on 2025-12-26) (cit. on p. 1).

[11] A. Dauda, O. Flauzac, and F. Nolot. "A Survey on IoT Application Architectures". In: *Sensors* 24.16 (2024). ISSN: 1424-8220. DOI: 10.3390/s24165320 (cit. on p. 1).

[12] C. J. Deepu, C.-H. Heng, and Y. Lian. "A Hybrid Data Compression Scheme for Power Reduction in Wireless Sensors for IoT". In: *IEEE Transactions on Biomedical Circuits and Systems* 11.2 (2017), pp. 245–254. DOI: 10.1109/TBCAS.2016.2591923 (cit. on pp. 20, 22).

[13] B. Dorsemaine et al. "Internet of Things: A Definition & Taxonomy". In: *2015 9th International Conference on Next Generation Mobile Applications, Services and Technologies*. 2015, pp. 72–77. DOI: 10.1109/NGMAST.2015.71 (cit. on p. 6).

[14] Espressif Systems. *ESP IoT Development Framework*. 2026. URL: https://www.espressif.com/en/products/sdks/esp-idf (visited on 2026-01-27) (cit. on p. 29).

[15] S. FakhrHosseini et al. "A taxonomy of home automation: expert perspectives on the future of smarter homes". In: *Information Systems Frontiers* 27.2 (2025), pp. 449–466. DOI: 10.1007/s10796-024-10496-9 (cit. on p. 6).

[16] P. Fouto et al. "Babel: A Framework for Developing Performant and Dependable Distributed Protocols". In: *2022 41st International Symposium on Reliable Distributed Systems (SRDS)*. 2022, pp. 146–155. DOI: 10.1109/SRDS55811.2022.00022 (cit. on pp. 3, 28).

[17] C. Geeng and F. Roesner. "Who's In Control? Interactions In Multi-User Smart Homes". In: *Proceedings of the 2019 CHI Conference on Human Factors in Computing Systems*. CHI '19. Glasgow, Scotland Uk: Association for Computing Machinery, 2019, pp. 1–13. ISBN: 9781450359702. DOI: 10.1145/3290605.3300498 (cit. on p. 6).

[18] S. Hamdan, M. Ayyash, and S. Almajali. "Edge-Computing Architectures for Internet of Things Applications: A Survey". In: *Sensors* 20.22 (2020). ISSN: 1424-8220. DOI: 10.3390/s20226441 (cit. on p. 1).

[19] K. Khanchuea and R. Siripokarpirom. "A Multi-Protocol IoT Gateway and WiFi/BLE Sensor Nodes for Smart Home and Building Automation: Design and Implementation". In: *2019 10th International Conference of Information and Communication Technology for Embedded Systems (IC-ICTES)*. 2019, pp. 1–6. DOI: 10.1109/ICTEmSys.2019.8695968 (cit. on pp. 10, 13, 15, 16).

[20] D. Kreković et al. "Reducing communication overhead in the IoT–edge–cloud continuum: A survey on protocols and data reduction strategies". In: *Internet of Things* 31 (2025), p. 101553. ISSN: 2542-6605. DOI: https://doi.org/10.1016/j.iot.2025.101553 (cit. on p. 1).

[21] J. Leitao, J. Pereira, and L. Rodrigues. "HyParView: A Membership Protocol for Reliable Gossip-Based Broadcast". In: *37th Annual IEEE/IFIP International Conference on Dependable Systems and Networks (DSN'07)*. 2007, pp. 419–429. DOI: 10.1109/DSN.2007.56 (cit. on pp. 10, 13, 18, 19, 32).

[22] H. Li, K. Ota, and M. Dong. "Always Connected Things: Building Disaster Resilience IoT Communications". In: *2019 IEEE 25th International Conference on Parallel and Distributed Systems (ICPADS)*. 2019, pp. 570–577. DOI: 10.1109/ICPADS47876.2019.00087 (cit. on pp. 14, 16).

[23] M5Stack. *ESP32 Core Basic*. 2026. URL: https://docs.m5stack.com/en/core/basic (visited on 2026-01-30) (cit. on p. 35).

[24] P. Maciel et al. "A survey on reliability and availability modeling of edge, fog, and cloud computing". In: *Journal of Reliable Intelligent Environments* 8.3 (2022), pp. 227–245 (cit. on pp. 1, 2).

[25] M. R. Mesbahi, A. M. Rahmani, and M. Hosseinzadeh. "Reliability and high availability in cloud computing environments: a reference roadmap". In: *Human-centric Computing and Information Sciences* 8.1 (2018), p. 20 (cit. on p. 1).

[26] R. Montella, M. Ruggieri, and S. Kosta. "A fast, secure, reliable, and resilient data transfer framework for pervasive IoT applications". In: *IEEE INFOCOM 2018 - IEEE Conference on Computer Communications Workshops (INFOCOM WKSHPS)*. 2018, pp. 710–715. DOI: 10.1109/INFCOMW.2018.8406884 (cit. on pp. 12, 13).

[27] J. Moy. *OSPF Version 2*. RFC 1247. 1991-07. DOI: 10.17487/RFC1247 (cit. on p. 25).

[28] J. Pan and J. McElhannon. "Future Edge Cloud and Edge Computing for Internet of Things Applications". In: *IEEE Internet of Things Journal* 5.1 (2018), pp. 439–449. DOI: 10.1109/JIOT.2017.2767608 (cit. on p. 2).

[29] J. Porte et al. "Heterogeneous wireless IoT architecture for natural disaster monitorization". In: *EURASIP Journal on Wireless Communications and Networking* 2020.1 (2020), p. 184 (cit. on pp. 11, 13, 14, 16).

[30] A. K. M. Al-Qurabat, C. Abou Jaoude, and A. K. Idrees. "Two Tier Data Reduction Technique for Reducing Data Transmission in IoT Sensors". In: *2019 15th International Wireless Communications & Mobile Computing Conference (IWCMC)*. 2019, pp. 168–173. DOI: 10.1109/IWCMC.2019.8766590 (cit. on pp. 22, 33).

[31] Raspberry Pi Ltd. *Raspberry Pi Pico SDK*. 2026. URL: https://www.raspberrypi.com/documentation/pico-sdk/index_doxygen.html (visited on 2026-01-27) (cit. on p. 30).

[32] J. Ren et al. “Collaborative Cloud and Edge Computing for Latency Minimization”. In: *IEEE Transactions on Vehicular Technology* 68.5 (2019), pp. 5031–5044. DOI: 10.1109/TVT.2019.2904244 (cit. on pp. 1, 2).

[33] K. Scott and S. C. Burleigh. *Bundle Protocol Specification*. RFC 5050. 2007-11. DOI: 10.17487/RFC5050 (cit. on p. 12).

[34] Z. Shelby, K. Hartke, and C. Bormann. *The Constrained Application Protocol (CoAP)*. RFC 7252. 2014-06. DOI: 10.17487/RFC7252 (cit. on p. 8).

[35] R. H. Shumway and D. S. Stoffer. “ARIMA Models”. In: *Time Series Analysis and Its Applications: With R Examples*. Cham: Springer International Publishing, 2017, pp. 75–163. ISBN: 978-3-319-52452-8. DOI: 10.1007/978-3-319-52452-8_3 (cit. on p. 20).

[36] S. Sinha. “State of IoT 2025: Number of connected IoT devices growing 14% to 21.1 billion globally”. In: *IoT Analytics* (2025) (cit. on p. 1).

[37] A. Staff. *Summary of the Amazon DynamoDB Service Disruption in the Northern Virginia (US-EAST-1) Region*. 2025. URL: https://aws.amazon.com/message/101925/ (visited on 2025-12-26) (cit. on p. 1).

[38] S. Suryavansh et al. “A data-driven approach to increasing the lifetime of IoT sensor nodes”. In: *Scientific Reports* 11.1 (2021), p. 22459. DOI: 10.1155/2018/4283087 (cit. on pp. 19, 22, 33).

[39] I. R. Team. *AWS Outage Analysis: October 20, 2025*. 2025. URL: https://www.thousandeyes.com/blog/aws-outage-analysis-october-20-2025 (visited on 2025-12-26) (cit. on p. 1).

[40] T. Theodorou et al. “A Multi-Protocol Software-Defined Networking Solution for the Internet of Things”. In: *IEEE Communications Magazine* 57.10 (2019), pp. 42–48. DOI: 10.1109/MCOM.001.1900056 (cit. on p. 16).

[41] S. Voulgaris, D. Gavidia, and M. Van Steen. “Cyclon: Inexpensive membership management for unstructured p2p overlays”. In: *Journal of Network and systems Management* 13.2 (2005), pp. 197–217 (cit. on p. 11).

[42] L. Xing. “Reliability in Internet of Things: Current Status and Future Perspectives”. In: *IEEE Internet of Things Journal* 7.8 (2020), pp. 6704–6721. DOI: 10.1109/JIOT.2020.2993216 (cit. on p. 2).

[43] N. Xiong et al. “Randomized and Efficient Time Synchronization in Dynamic Wireless Sensor Networks: A Gossip-Consensus-Based Approach”. In: *Complexity* 2018.1 (2018), p. 4283087. DOI: 10.1155/2018/4283087 (cit. on pp. 17, 18, 33).

[44] W. Yu et al. “A Survey on the Edge Computing for the Internet of Things”. In: *IEEE Access* 6 (2018), pp. 6900–6919. DOI: 10.1109/ACCESS.2017.2778504 (cit. on p. 1).