

**Comparing JSON and ProtoBuf in HTTP-based REST architectures: performance and energy efficiency**

**Miguel Alves Ferreira**

**Preparação para Dissertação**

**Engenharia Informática**

**Área de Especialização em Engenharia de software**

Integrity Statement

I declare that I have conducted this academic work with integrity.

I have not plagiarized or applied any form of misuse of information or falsification of results throughout the process that led to its elaboration.

Therefore, the work presented in this document is original and of my authorship and has not been used previously for any other purpose.

I also declare that I am fully aware of P.PORTO’s Code of Ethical Conduct.

ISEP, Porto, 7 de fevereiro de 2025

Summary

O objetivo principal deste documento de preparação da dissertação é formular um plano para a dissertação, que envolve uma análise detalhada dos trade-offs entre JSON (Javascript Object Notation) e Protocol Buffers em arquitecturas REST (Representational State Transfer). Ambos os formatos de serialização são bem conhecidos na indústria de software, sendo o JSON um formato mais simples e legível, mas que apresenta ineficiências quando utilizado em larga escala. Por outro lado, o ProtoBufs (Protocol Buffers), um formato de serialização binário desenvolvido pela Google, apresenta vantagens como a redução do tamanho, o aumento do desempenho e a redução da utilização de memória em comparação com o JSON. No entanto, a sua natureza binária torna-o menos legível.

O documento em análise contém o *project charter*, que aborda o âmbito do projeto, descreve o objetivo do projeto, identifica as partes interessadas envolvidas, enumera os benefícios que o projeto trará, delineia os entregaveis futuros, estima os custos relacionados com o projeto, estabelece pressupostos e restrições e identifica possíveis riscos associados ao projeto.

Para além disso o estudo utilizará uma metodologia de experiência controlada para analisar diferentes métricas que comparam igualmente JSON e Protocol Buffers, neste caso ambos aplicados em arquitecturas REST.

**Palavras-chave**: JSON, Protocol Buffers, REST, serialização, desempenho, eficiência

Abstract

The dissertation preparation document outlines a plan to analyze trade-offs between JSON and Protocol Buffers in REST architectures. JSON, while simpler and more readable, is less efficient at scale compared to Protocol Buffers, which offer smaller sizes, better performance, and lower memory usage but lack readability due to their binary nature. The document includes a project charter detailing the project's scope, objectives, stakeholders, benefits, deliverables, costs, assumptions, constraints, and risks. The study will use a controlled experiment methodology to compare metrics of both serialization formats in the context of REST architectures.

**Keywords**: JSON, Protocol Buffers, REST, serialization, performance, efficiency

Acknowledgements

During high school, I never imagined that I would go to university. I was a poor student, lazy and lacking any vision for my future. But that changed because of Marcia Pacheco, an incredible biology teacher who never gave up on any student, including me. For that, I am deeply grateful to her for everything she has done for me.

At the start of my computer science degree, I felt demotivated. I had initially wanted to study biology, but I couldn’t because of my marks, so I chose computer science instead. Additionally, I wasn’t enjoying the first year of university. My switch from a public institution to a private one felt like a step back, as I saw private universities as less prestigious. However, I couldn’t have been more wrong. The first semester was challenging and a harsh reality check, but things began to change after that.

In my second semester, I had the privilege of learning from Paula Morais, a teacher who reignited my passion for technology. She taught me how to program, answered even the most basic questions with patience, and inspired me to delve deeper into the field. She was not just a great teacher but also an approachable and supportive mentor. I am immensely grateful to her for helping me rediscover my love for technology and encouraging me to continue pursuing my studies.

I would also like to thank Fátima Lopes for motivating me to enroll in a master’s degree program, an opportunity I would likely never have pursued without her encouragement.

Finally, and most importantly, I want to express my heartfelt gratitude to my family. Their unwavering support, understanding, and encouragement have been invaluable. They have always respected my priorities and never criticized the time I devoted to projects, assignments, and exams. Their belief in me gave me the confidence to complete my computer science degree and continue to my master’s.

Without these incredible people, I would not be where I am today, nor would I be the person I have become.

Index

1 Introduction 1

1.1 Background and Context 1

1.2 Problem Description 2

1.3 Objectives and Research Questions 2

1.4 Ethical Considerations 3

1.5 Report Structure 3

2 Background 4

2.1 REST 4

2.1.1 How are REST Applications Structured 6

2.2 JSON 7

2.3 Protocol Buffers 9

2.3.1 Structure of Protocol Buffers 9

2.3.2 Schema Compilation and Code Generation 10

2.4 Performance and Energy Analysis Tools 11

2.4.1 JMeter 11

2.4.2 Kepler 11

3 Planning 13

3.1 Project Charter 13

3.1.1 Stakeholders 13

3.1.2 Scope 14

3.1.3 Objectives 14

3.1.4 Benefits 15

3.1.5 Deliverables 15

3.1.6 Time 15

3.1.7 Costs 16

3.1.8 Assumptions and Restrictions 16

3.1.9 Risk 17

3.2 Work Breakdown Structure 17

3.3 Work Breakdown Structure Dictionary 18

3.4 Timeline 21

3.5 Skills 22

3.5.1 Skills Required for the Project 22

3.5.2 Strengths 22

3.5.3 Weaknesses 23

3.5.4 Technical Skills 23

3.5.5 Plans for Improvement 25

4 Literature Review 26

4.1 Research Questions 26

4.2 Data Sources 26

4.3 Search Terms 27

4.4 Eligibility Criteria 28

4.5 Data Collection Process 28

4.6 Discussion 29

4.6.1 RQ1: How does using ProtoBufs impact the performance of HTTP-based REST architectures? 29

4.6.2 RQ2: To what extent can ProtoBufs improve energy efficiency in HTTP-based REST architecture communications compared to JSON? 34

4.7 Conclusion 35

5 Research Methodology 37

5.1 Why Controlled Experiment Methodology? 37

5.2 Data Collection 37

List of Figures

[Figure 1 - REST API in Action image taken from [16] 5](#_Toc186891431)

[Figure 2 - Overall compliance with the REST principles of 500 REST APIS taken from[9] 6](#_Toc186891432)

[Figure 3 – Timeline part 1/2 22](#_Toc186891433)

[Figure 4 – Timeline part 2/2 22](#_Toc186891434)

[Figure 5 - PRISMA Systematic Methodology altered from [32] 29](#_Toc186891435)

[Figure 6 - Average serialization time taken from [1] 31](#_Toc186891436)

[Figure 7 - Average Deserialization time taken from [1] 31](#_Toc186891437)

[Figure 8 – Response time benchmark of a GET request uncompressed taken from [22] 32](#_Toc186891438)

[Figure 9 – Payload size benchmark with flat data taken from [22] 33](#_Toc186891439)

[Figure 10 – Memory analysis of JSON serialization taken from [22] 33](#_Toc186891440)

[Figure 11 – Memory analysis of Protocol Buffers serialization taken from [22] 34](#_Toc186891441)

[Figure 12 – Energy expended on CPU with data synchronization (volume1) taken from [7] 35](#_Toc186891442)

[Figure 13 – Energy expended on CPU with data synchronization (volume2) taken from [7] 35](#_Toc186891443)

[Figure 14 – Work Breakdown Structure 43](#_Toc186891444)

[Figure 15 – Full timeline 44](#_Toc186891445)

[Figure 16 – Improved resolution of timeline part 1/2 44](#_Toc186891446)

[Figure 17 – Improved resolution of timeline part 2/2 45](#_Toc186891447)

[Figure 18 – Risk Register Part 1 46](#_Toc186891448)

[Figure 19 – Risk Register Part 2 47](#_Toc186891449)

List of Tables

Table 1 – List of Types that can be defined in the *.proto* file 10

Table 2 – Power and interest matrix 13

Table 3 – WBS dictionary 18

Table 4 – Technical Skills needed to complete the dissertation 24

Table 5 – Data sources 27

Table 6 - Average Serialization Time in ms taken from [1] 30

Table 7 - Average Serialization Time in ms taken from [1] 30

Table 8 – “A list of space-efﬁciency benchmark publications that involve JSON[…]” [2] altered from [2] 50

List of Code Snippets

[Code Snippet 1 – Example of a User JSON 7](#_Toc186877457)

[Code Snippet 2 – Example of Decoding and Encoding a User JSON in Golang 8](#_Toc186877458)

[Code Snippet 3 – Block of code from a *.proto* file defining a User 9](#_Toc186877459)

[Code Snippet 4 – Search query developed with the search terms and research questions 27](#_Toc186877460)

Acronyms and Symbols

**Acronym List**

**JSON** *Javascript Object Notation*

**IDL** *Interface Definition Language*

**XML** *Extensible Markup Language*

**REST** *Representational State Transfer*

**HATOAS** *Hypermedia As The Engine Of Application State*

**URI** *Uniform Resource Identifier*

**gRPC** *Google Remote Procedure Call*

**IoT** *Internet Of Things*

**HTTP** *Hypertext Transfer Protocol*

**API** *Application Programming Interface*

**Go** *Golang*

**Kepler** *Kubernetes Efficient Power Level Exporter*

**eBPF** *extended Berkeley Packet Filter*

# Introduction

The proceeding chapter will detail the background, and the context and describe the problem, it will also depict the objectives of the research being conducted, and it will also include the ethical considerations about the continuous development of the dissertation. Finally, it will also encompass the structure of the report.

## Background and Context

In the ever-evolving field of computer science and software engineering, efficient and scalable communication protocols are paramount. The rise of cloud computing, IoT (Internet of Things), and microservices has led to a proliferation of REST (Representational State Transfer) APIs (Application Programming interfaces), which rely heavily on serialization formats like JSON (JavaScript Object Notation), XML (Extensible Markup Language), ProtoBufs (Protocol Buffers) and others. Among these, JSON has emerged as the de facto standard due to its simplicity, human readability, and wide support across programming languages. However, JSON’s text-based, schema-less nature results in larger payloads, slower parsing times, and increased network overhead, especially for large datasets of data [1], [2], [3], [4], [5], [6]. In contrast, ProtoBufs is a binary serialization format developed by Google that offers a more efficient, compact, and schema-driven alternative to JSON. ProtoBufs’ binary encoding reduces payload sizes, and improves parsing speed, making it ideal for high-performance applications with stringent resource constraints [4], [7], [8]. Despite these advantages, ProtoBufs' complexity and lack of human readability might limit its adoption in REST architectures, where JSON remains the dominant choice.[9]. This study aims to evaluate the trade-offs between JSON and ProtoBufs in REST applications, focusing on performance, scalability, resource consumption, and energy efficiency to provide insights for developers and software architects seeking to optimize their communications in REST applications.

## Problem Description

Choosing the right technology for serializing data plays a major role in the performance and scalability of REST applications, they will be with it from start to finish of the project [10]. JSON the most popular serialization format used because of its simplicity, human readability, and support across programming languages has drawbacks that could be improved by using Protocol Buffers in the same REST application. Protocol Buffers, exceptionally used with gRPC (Google Remote Procedure Call) allows for a more efficient, compact, and schema-driven alternative to JSON, but the complexity of implementing or migrating to gRPC is considerably higher than implementing a REST architecture. Performance and energy efficiency are increasingly critical in software development, particularly in HTTP-based REST architecture as such, increased latency, reduced throughput, or inefficient resource usage can significantly impact user experience and system scalability. Research shows that underutilized or poorly optimized systems lead to higher costs, resource wastage, and scalability issues [11], [12]. For this reason, efficient serialization formats like ProtoBufs could reduce these bottlenecks by minimizing payload sizes and improving processing speed. As such, energy efficiency is another growing concern, especially in environments like mobile devices, IoT, and data centers. Research has shown that energy consumption impacts operational costs and plays an important role in system design decisions and the sustainability of large-scale deployments [12], [13]. In modern data centers, serialization optimizations can lead to substantial energy savings, translating into lower costs and reduced environmental impact [12].

## Objectives and Research Questions

The Primary objective of the research is to analyze and compare the performance and energy efficiency of ProtoBufs and JSON in HTTP-based REST architectures.

Research Questions:

RQ1: How does using ProtoBufs impact the performance of HTTP-based REST architectures compared to JSON?

RQ2: To what extent can ProtoBufs improve energy efficiency in HTTP-based REST architecture communications compared to JSON?

## Ethical Considerations

This study will adhere to ethical principles to ensure integrity and transparency. Although the research will not involve human subjects or sensitive data related to any possible subjects, the study will firmly comply with the following ethical aspects:

1. Data Privacy and Security: All test data to be used in the study will be synthetic and generated specifically for benchmarking purposes. No real user data or proprietary information will be used, ensuring compliance with data protection regulations such as RGPD (Regulamento Geral de proteção de dados).
2. Transparency and Reproducibility: The experimental setup, tools, and methodologies are going to be documented in detail to allow reproducibility of the results by other researchers. Open-source tools and datasets will be used whenever possible [14].
3. Benchmarking Practices: There will be extreme care to avoid bias in performance evaluations. Both serialization formats will be tested under equal conditions, whenever possible, to ensure fair comparisons.

## Report Structure

* Introduction: Provides context, problem definition, objectives, and ethical considerations.
* Background: Introduces Information about the tools being studied like JSON, Protocol Buffers, REST, and metric analysis tools.
* Planning: This shows the Project charter in detail, it also includes the WBS (Work Breakdown Structure), the WBS dictionary, the timeline, and finally the skills needed to complete the project successfully.
* Literature Review: Summarizes current findings on serialization formats and their possible connection to REST applications.
* Research Methodology: Details the experimental methodology Being used.

# Background

The following section will provide an overview of fundamental concepts related to REST, JSON, and ProtoBufs, setting the stage for the subsequent chapters.

## REST

Representational State Transfer (REST) is an architectural style introduced in 2000 [15]. It defines a set of standards and constraints for designing distributed systems, especially web services. It also provides a lightweight stateless client-server communication model that can enable an application to be scalable, efficient, and easy to maintain. These factors contribute to its wide usage. Besides, simplicity and compatibility with web protocols make it adaptable to a wide variety of clients, from web browsers and mobile phones to complex IoT systems. REST principles are foundational in contemporary architecture such as microservices and cloud computing [16].

A diagram of a rest api

Description automatically generated

Figure 1 - REST API in Action image taken from [16]

At its core, REST leverages HTTP methods like GET, POST, PUT, DELETE, and PATCH to perform operations on resources. These methods correspond to common CRUD (Create, Read, Update, Delete) operations:

* GET: Retrieves a resource or collection of resources.
* POST: Creates a new resource.
* PUT: Updates an existing resource or creates it, if it doesn’t exist.
* DELETE: Removes a resource.
* PATCH: Partially updates a resource.

Data exchanged in RESTful systems can be serialized in various formats, such as JSON, XML, or ProtoBufs, this flexible design allows developers to choose the most suitable format for their specific use case. Arguably REST offers significant benefits such as scalability, flexibility, and low maintenance costs, but it has some limitations. RESTful APIs can suffer from:

* High Latency: Multiple client-server interactions may increase response times.
* High Bandwidth Usage: Particularly when verbose formats like JSON or XML are used.
* High Energy Consumption: Significant processing power may be required for encoding, decoding, and data transfer.

### How are REST Applications Structured

Restful applications follow a well-defined structure and set of principles to ensure scalability, maintainability, and interoperability. These principles are summarized as REST constraints, which define how REST APIS should behave [17], [18]:

1. Statelessness: Every client-server interaction must be independent. The server does not store any information about previous requests; instead, each request must contain all the necessary information to process it. This ensures scalability and simplifies server implementation.
2. Client-Server Architecture: The client and server are separate entities, allowing them to evolve independently. The client is responsible for the user interface and request initiation, while the server handles data storage and business logic.
3. Uniform Interface: Rest applications must follow a standardized way of interacting with resources. This includes:

* Resource Identification which is done through URIs (Uniform Resource Identifiers), For example, */users/1 identifies* a specific user resource.
* Representation of resources are represented in formats like JSON, XML, or ProtoBufs.
* HATEOAS (Hypermedia as The Engine of Application State) is the principle that clients should navigate the application through hyperlinks embedded in responses. This allows for the dynamic discovery of resources and actions.

1. Layered System: REST applications can be designed with multiple layers (e.g., caching servers, authentication layers, load balancers) that operate independently. The client interacts with the server as if it were a single entity, without being aware of intermediate layers.
2. Cacheability: Responses from the server should explicitly indicate whether they are cacheable. The proper use of caching can reduce latency and bandwidth usage, improving overall performance.
3. Code on Demand: In essence allows for the server to send executable code to the client.

A graph with numbers and a bar

Description automatically generated

Figure 2 - Overall compliance with the REST principles of 500 REST APIS taken from[9]

Although REST has become the dominant architectural style for web services, studies show that many implementations deviate significantly from its core principles as we can see in *Figure 2*. For instance, a large-scale analysis of 500 public REST APIs revealed that only 0.8% of the services fully conformed to all REST architectural principles [9].

## JSON

JSON (JavaScript Object Notation) was introduced in 2001 and has become one of the most widely used data serialization formats in web and application development. It was created to provide a lightweight, easy-to-parse format for transmitting structured data. Designed with simplicity and efficiency in mind, JSON has become integral to modern software systems that require real-time data exchange with minimal overhead [19].

At its core, JSON represents data as key-value pairs. Keys are always strings, while values can be of various data types, including strings, numbers, booleans, arrays, objects, or null. JSON's syntax is intuitive and precise, adhering to the following rules:

* Keys must be enclosed in double quotes (").
* Objects are encapsulated within curly braces ({}) and consist of key-value pairs separated by colons (:).
* Arrays are enclosed in square brackets ([]) and represent ordered lists of values.
* Values can include primitive data types (e.g., strings or numbers) or complex structures like nested objects and arrays.

Below is an example of a JSON object that demonstrates these principles:

{

"name": "John Doe",

"age": 30,

"address": {

"street":

"123 Main St",

"city": "Anytown",

},

"phone": [

"123-456-7890",

"987-654-3210"

]

}

Code Snippet 1 – Example of a User JSON

In this example, the *name* and *age* fields represent simple string and numeric values, respectively, while the *address* field is a nested object containing its key-value pairs. The *phone* file demonstrates the use of arrays to store multiple items. Furthermore, the flexibility of JSON also provides for the creation of complex data structures which makes it great for most applications.

Additionally, JSON's popularity stems from its simplicity, versatility, and widespread use in various environments. It is extensively used in APIs to enable seamless data exchange between servers and clients. It also appears in configuration files for storing application settings and in real-time communication systems, such as chat applications and IoT devices, where lightweight and efficient data representation is essential [1], [7], [8]. Another key feature of JSON is its ease of parsing and generation. Most modern programming languages provide built-in libraries or modules to handle JSON, allowing developers to effortlessly serialize and deserialize data. For example, in Go (Golang), JSON encoding and decoding are facilitated by the encoding/json package. The following example demonstrates how the encoding and decoding of JSON data can be achieved in Go (Golang) with the previously defined JSON object:

package main

import (

"encoding/json"

"fmt"

)

type Person Struct {

Name string `json: "name"`,

age int `json: "age"`,

Address struct{

Street string `json:"street"`,

City string `json:"city"`,

}

Phone []string `json:"phone"`,

}

func main() {

jsonData := `{

"name": "John Doe",

"age": 30,

"address": {

"street": "123 Main St",

"city": "Anytown",

},

"phone": ["123-456-7890", "987-654-3210"]

}`

var person Person

err := json.Unmarshal([]byte(jsonData), &person) // Decoding JSON data into Go struct

if err != nil {

fmt.Println(err)

return

}

fmt.Printf("Decoded Struct: %+v\n", person)

encodedData, err := json.Marshal(person) // Encoding Go struct into JSON data

if err != nil {

fmt.Println("Error encoding JSON:", err)

return

}

fmt.Println("encode JSON: ", string(encodedData))

}

Code Snippet 2 – Example of Decoding and Encoding a User JSON in Golang

In this Go example, the *Person* *struct* defines the expected structure of the JSON object. The *json.Unmarshal* function is used to parse JSON into this struct, while *json.Marshal* converts the struct back into a JSON string. This demonstrates JSON's ability to integrate seamlessly into strongly typed programming while maintaining simplicity and efficiency.

## Protocol Buffers

Protocol Buffers, commonly referred to as ProtoBufs, were initially developed internally at Google in 2001 and released to the public in 2008. According to the official documentation, "Protocol buffers are language-neutral, platform-neutral extensible mechanisms for serializing structured data" [20]. ProtoBufs is fundamentally a binary serialization format that facilitates efficient and compact data exchange between applications. Its design prioritizes performance, compactness, and simplicity, making it an ideal choice for scenarios where high throughput and low latency are critical [6], [8], [21], [22].

ProtoBufs are extensively used in applications such as microservices, distributed systems, and Internet of Things (IoT) devices, where efficiency in data communication is paramount. A prominent use case is its integration with gRPC, a high-performance framework for inter-service communication, which leverages ProtoBufs for serializing and deserializing messages. This combination enables developers to build scalable and efficient APIs and distributed systems[23], [24]. While ProtoBufs are often noted for outperforming other serialization formats like XML and JSON In terms of speed and data compactness, comparisons must consider architectural differences. For instance, evaluations frequently compare ProtoBufs in gRPC-based architectures against JSON in RESTful architectures[3], [4], [24]. Such comparisons may introduce variability in the results of certain metrics due to the intrinsic differences between the two paradigms, such as transport protocol overhead and communication patterns.

### Structure of Protocol Buffers

Protocol Buffers use a schema-based serialization approach, meaning the data structure must be explicitly defined in a schema file with the *.proto* extension. This schema, written in ProtoBufs Interface Definition Language (IDL), outlines the data structure, specifying field names, types, and nested structures. The schema also includes metadata, such as the syntax version, which defaults to Proto2 unless explicitly specified as Proto3.

The following is an example of a *.proto* file defining message types for a Person and their Address:

syntax = "proto3"; // Here is the version of the syntax if omitted proto2 is assumed

message Person {

string name = 1;

int32 id = 2;

Address address = 3;

repeated string phone = 4;

}

message Address {

string street = 1;

string city = 2;

}

Code Snippet 3 – Block of code from a *.proto* file defining a User

In the Schema:

* Each Field is assigned a unique number, used as a field identifier in the binary encoding.
* The repeated keyword indicates that a field can hold multiple values, effectively defining a list or array.
* Nested message types such as Address within Person, allow for the definition of hierarchical data structures.

ProtoBufs support a rich set of data types to address diverse use cases. Table 1 summarizes the core data types supported by ProtoBufs:

Table 1 – List of Types that can be defined in the *.proto* file

| **Type** | **Description** |
| --- | --- |
| double | Double Precision floating point |
| float | Single precision floating point |
| int32 | Signed 32-bit integer |
| int64 | Signed 64-bit integer |
| uint32 | Unsigned 32-bit integer |
| uint64 | Unsigned 64-bit integer |
| sint32 | Signed variable-length integer |
| sint64 | Signed variable-length integer |
| bool | Boolean value |
| string | String |
| bytes | Bytes |
| fixed32 | 32-bit fixed-point |
| fixed64 | 64-bit fixed point |
| sfixed32 | 32-bit signed fixed-point |
| sfixed64 | 64-bit signed fixed-points |

The table depicted above shows the types that ProtoBufs supports.

### Schema Compilation and Code Generation

Once the *.proto* schema file is created, it must be compiled using the *protoc* compiler, which generates language-specific code based on the schema definition. This generated code provides data structures and methods for encoding and decoding ProtoBufs messages. For example, in Go (Golang), the *protoc* compiler produces Go structs and associated methods for interacting with the data. These files typically have a *.pb.go* extension in Go projects.

## Performance and Energy Analysis Tools

Using performance and energy analysis tools is imperative for assessing the efficiency and resource utilization of software systems. These tools allow us to observe applications under diverse workloads and operational conditions. In this dissertation, two analytical tools are employed: JMeter and Kepler.

### JMeter

Apache JMeter is a versatile, open-source performance testing tool developed to evaluate the performance and reliability of web applications and services. Simulating concurrent user requests to a target server enables developers and testers to analyze system behaviour under variable load conditions. Its support for multiple protocols, including HTTP, HTTPS, and FTP, enhances its utility as a comprehensive performance-testing solution.

**Main Types of Tests:**

1. **Load Test**: “Load testing focuses on the ability of a system to handle increasing levels of anticipated realistic loads resulting from transaction requests generated by controlled numbers of concurrent users or processes.” [25]
2. **Stress Test**: “Stress testing focuses on the ability of a system or component to handle peak loads that are at or beyond the limits of its anticipated or specified workloads. Stress testing is also used to evaluate a system’s ability to handle reduced availability of resources such as accessible computing capacity, available bandwidth, and memory.” [25]
3. **Endurance Test**: “Endurance testing focuses on the stability of the system over a time frame specific to the system’s operational context. This type of testing verifies that there are no resource capacity problems (e.g., memory leaks, database connections, thread pools) that may eventually degrade performance and/or cause failures at breaking points.” [25]
4. **Spike Test**: “Spike testing focuses on the ability of a system to respond correctly to sudden bursts of peak loads and return afterwards to a steady state.” [25]

### Kepler

Kepler (Kubernetes Efficient Power Level Exporter) is an open-source tool designed to monitor and estimate the energy consumption of Kubernetes workloads. By leveraging technologies like eBPF (extended Berkeley Packet Filter) and machine learning models, Kepler provides detailed insights into power usage at the process, container, and pod levels, facilitating more energy-efficient operations within Kubernetes environments. Kepler also provides quite a few features like [26], [27]:

* **Data collection via eBPF and Hardware Counters:** This allows Kepler to directly collect performance metrics from the Linux kernel and due to utilizing hardware counters, it is also capable of gathering detailed information on energy consumption estimates [26], [27].
* **Real-time power consumption metrics:** Kepler can access data from hardware components through their API to get accurate and real-time power metrics [26], [27].
* **Power consumption attribution:** Kepler applies the ratio power model to attribute power usage to individual processes. This model calculates the proportion of a process’s resource utilization relative to the entire system and multiplied by the dynamic power consumption of the resource. As such, by mapping process IDs to container and pod identifiers, Kepler can aggregate power consumption metrics, providing insight at various levels of the Kubernetes hierarchy.

# Planning

The current chapter will describe the project's planning, which will be crucial to finishing the project on time and having palpable dates to define what has already been done and what needs to be done.

## Project Charter

The project charter is a living document that should represent the brief planning of the project.

### Stakeholders

This project provides valuable insights for diverse stakeholders, who may benefit in various ways from the knowledge presented. Table 2 outlines the defined stakeholders, which include these stakeholders.

Table 2 – Power and interest matrix

| **Name** | **Power** | **Interest** |
| --- | --- | --- |
| Developers | Low | Medium |
| Software development companies | Medium | High |
| Students | Low | Medium |
| Researchers | High | High |
| Advisor | High | High |

Stakeholder Roles:

* Developers: Developers benefit from the project’s insights into serialization technologies, enabling them to make better technical decisions
* Software Development Companies: These companies aim to reduce costs and improve performance. The findings may influence their decisions about adopting Protocol Buffers within REST architecture.
* Students: The project will serve as a valuable learning resource, helping students learn more about serialization and the impact it has on a software system.
* Researchers: The findings may inspire further academic studies.
* Advisor: The advisor will ensure the project aligns with academic standards, providing guidance and assistance in overcoming challenges.

### Scope

Nowadays in software engineering, the efficiency of data serialization has a direct influence on the performance of REST architectures, JSON, as the dominant serialization format, is highly regarded for its simplicity and human readability. However, it has some performance issues, such as larger payloads and slower parsing, making it less suitable for resource-constrained or high-throughput scenarios. On the other hand, Protocol Buffers (ProtoBufs) is a schema-driven binary serialization format, that offers significant performance advantages like reduced payload sizes, improved parsing speeds, and optimized resource consumption. Despite these benefits, this serialization format has increased complexity and isn’t human-readable, and as such, it creates a barrier to its adoption.

With the increased demand for scalable systems, information like energy efficiency, and high performance, are crucial for microservices and IoT applications, as such, obtaining insights on how serialization formats might influence these aspects in a system is important. Furthermore, the research explores the trade-offs between JSON and ProtoBufs within REST architectures, additionally, it analyses key metrics, such as latency, resource usage, and energy efficiency giving the necessary insights to optimize communication in software systems.

### Objectives

The primary objective of this thesis is to perform a comprehensive evaluation of Protocol Buffers in REST architectures and compare key metrics against JSON, these metrics are:

* **Performance**: Analyze the latency and throughput for both formats.
* **Scalability**: See how well each format performs under different loads and data sizes.
* **Energy Consumption**: View the CPU energy utilization during serialization and deserialization.
* **Resource Efficiency**: Compare memory utilization and processing overhead.

These insights are intended to assist developers and architects with actionable advice on whether to use Protocol Buffers in RESTful apps.

### Benefits

The benefits the current research will provide are:

* **Cost Reduction:** If ProtoBufs demonstrates lower energy consumption or smaller payloads, this could translate into reduced bandwidth and operational costs.
* **Improved Application Performance:** Faster serialization and smaller payload sizes improve response times, which is critical for performance-sensitive applications.
* **Minimized Refactoring:** By proving the viability of ProtoBufs in REST architectures, companies may avoid fully transitioning to gRPC, saving resources while retaining REST’s simplicity.
* **Advancements in Software Development:** The research could encourage broader adoption of ProtoBufs and inspire innovation in serialization methods.
* **Scalability Insights:** Findings will help companies and developers better scale their systems by understanding serialization impacts.

### Deliverables

The project aims to produce the following deliverables:

1. **Project Plan:** Detailed timeline and task breakdown.
2. **Dissertation Report:** A thesis that documents the study, techniques, findings, and interpretations.
3. **Software Code:** All Code related to the experimental setup, including benchmarks and test scripts.
4. **Testing Reports:** Detailed analysis of performance metrics such as latency, throughput, and energy consumption.
5. **Data Collected:** Processed and raw data from benchmark experiments.
6. **Presentation and Discussion:** A final presentation summarizing the project’s key findings, to be delivered to stakeholders.

### Time

These are the following milestones and mandatory dates for the project. The milestones can and probably will change due to my lack of knowledge of the processes, but the mandatory dates will hopefully stay the same throughout the project.

**Milestones:**

* GQM (Goal Question Metric) – 15/01/2025
* Hypothesis Test – 24/01/2025
* Control Project – 13/02/2025
* Data From Control Project – 14/03/2025
* Changed Project – 14/03/2025
* Data from changed project – 09/04/2025
* Finished analysis – 09/04/2025
* Report 1.0 delivery – 17/04/2025
* Final Report Delivery – 30/04/2025
* Presentation Date – 26/06/2025

**Mandatory Dates:**

* Prepd review delivery – 06/12/2024
* Report and Presentation Delivery – 04/01/2024
* Final Delivery – 26/06/2024

### Costs

No direct costs have been identified for this project, as it relies on open-source tools (e.g., JMeter, and Kepler which were already presented before in the background section), institution-provided infrastructure, and digital libraries accessible through academic licensing. However, indirect costs such as time investment and a learning curve for new tools and methodologies are expected. These are managed by scheduling ample time for setup and practice.

### Assumptions and Restrictions

**Assumptions:**

1. Benchmarking tools and required libraries will be stable and accessible throughout the project.
2. Synthetic data will adequately simulate real-world scenarios.
3. Experimental environments will be consistent across tests.
4. Digital libraries for literature review will remain accessible.

**Restrictions**

1. Access to real-world datasets is limited, the study relies on synthetic data for experiments.
2. Resource limitations may restrict the scale of experiments (e.g. server capacity or cloud computing costs).
3. Benchmarking tools may impose constraints on the range of metrics that can be measured (e.g. some tools may not support energy profiling).
4. Tests are conducted in controlled environments, which might not reflect all real-world network scenarios.
5. A fixed timeline may limit the number of experiments done.

### Risk

Risk management is going to be done to ensure that potential threats are identified, assessed, and mitigated, this should ensure that there won’t be any disruptions or compromises in the project timeline, consequently, creating a Risk Register was a top priority to document and monitor risks throughout the project lifecycle, it identifies key risks, their causes, and the planned responses.

The Risk Register highlights 9 primary risks (Figures 18 and 19 in Appendix A), covering various aspects of the project, including tool compatibility, data quality, and resource availability. Each risk has been evaluated for its probability and impact, resulting in a prioritization based on the PI score. Below are some examples:

* **High-Priority Risks**:
  + Synthetic data bias (PI Score: 15)
  + Time constraints limiting experimental scope (PI Score: 15)
* **Medium-Priority Risks**:
  + Tool incompatibility (PI Score: 8)
  + Data loss due to accidental corruption (PI Score: 10)

Mitigation strategies have been outlined for each risk to ensure project continuity and success. The Risk Register is a living document that will be reviewed and updated periodically as new risks emerge or existing risks evolve.

## Work Breakdown Structure

The work Breakdown structure is in Appendix A figure 14, and it is divided into 3 main phases. The experimental Design and Setup involves defining the research methodology, establishing the experimental environment, and collecting baseline data. It sets the foundation for the comparison between JSON and ProtoBufs, the Documentation and Reporting phase, and data from the experiments are analyzed to identify trends, draw conclusions, and address the research questions, the findings are then documented in a structured format to form the core of the dissertation, finally, we have the final phase focusing on refining the dissertation based on feedback, preparing the presentation, and ensuring all deliverables are submitted on time. It concludes with the final presentation and evaluation of the research.

## Work Breakdown Structure Dictionary

This section provides a comprehensive examination of the detailed WBS (work breakdown structure) dictionary. The following table is a dictionary in which each row corresponds to a phase, deliverable or work package. Each entry is accompanied by a description and progress criteria. Empty descriptions are self-explanatory and require no further elaboration.

Table 3 – WBS dictionary

|  |  |  |  |
| --- | --- | --- | --- |
| **WBS dictionary** | **Type** | **Description** | **Progress criteria** |
| 1 Experimental design and setup | Phase | This phase will mainly comprise setting up the projects and acquiring data for analysis. | Experimental methodology – 25%  Control project – 25%  Metrics analysis – 10%  Project refactor – 25%  Metrics analysis – 10%  Conclusions – 5% |
| 1.1 Experimental methodology | Deliverable | The variables, GQM plan and hypothesis must be defined to have a default execution for each analysis.  The advisor must accept the defined variables, GQM plan and hypothesis | Define variables to be analyzed – 15%  Define measurement methods -10%  Create GQM plan – 20%  Identify tools and technologies – 5%  Define Hypothesis – 15%  Acceptance from advisor – 30% |
| 1.1.1 Do the two LinkedIn leadership courses | Work package |  | Finish the course – 100% |
| 1.1.2 Define variables metrics and measurement methods | Work package | In this work package, the independent and dependent variables should be defined. The metrics to be analyzed and the measurement methods should also be defined. | Independent variables – 25%  Dependent variables – 25%  Define Metrics – 25%  Create a standardized Dataset – 5%  Review and verify tooling – 20% |
| 1.1.3 Create a detailed GQM plan | Work package | The Goal, questions and metrics should be defined | Goal – 35%  Questions – 25%  Metrics – 40% |
| 1.1.4 Identify tools and technologies | Work package | Tools and technologies should already be defined, but this work package should review the tools used and anticipate if new tools are needed. | Review tools – 50%  Obtain new tools – 50% |
| 1.1.5 Formulate hypothesis testing | Work package |  | Hypothesis defined – 40%  Choose a statistical method – 30%  Review and define experiment conditions -30% |
| 1.1.6 Report version 0.1 | Work package | Update the dissertation document with the work done in the 1.1 deliverable and review the literature. | Updated document – 100% |
| 1.2 Control project | Deliverable | Acquire and refactor control project, | Acquire control project – 30%  Refactor needed parts – 10%  Setup monitoring tools – 30%  Setup test environment – 30% |
| 1.2.1 Do a project risk course on Linked | Work package |  | Finish the course – 100% |
| 1.2.2 Acquire control project | Work package | Acquire a project compatible with the tooling defined previously | Find project – 100% |
| 1.2.3 Refactor needed parts | Work package | Refactor parts of the project that need to be changed | Define refactoring goals – 10%  Implement the refactoring -40%  Test refactored parts – 50% |
| 1.2.4 Setup monitoring tools | Work package | Install and configure defined monitoring tools | Install monitoring tools – 50%  Configure monitoring tools – 50% |
| 1.2.5 Setup test environment | Work package | In this work package, infrastructure and environment configuration should be provided. | Provision infrastructure – 50%  Configure environment – 50% |
| 1.2.6 Report version 0.2 | Work package | Update the dissertation document with the work done in the 1.1 deliverable and review the literature. | Updated document – 100% |
| 1.3 Metric analysis | Deliverable | This deliverable should have the data collected from the control project and conclusions about that data. | Collect data – 50%  Analysis of the data – 25%  Conclusion about the data – 25% |
| 1.3.1 Collect data | Work package |  | Collect data – 100% |
| 1.3.2 Make conclusions about the data collected | Work package |  | Analysis of the data-30%  Conclusion about the data-60%  Update document – 10% |
| 1.4 Project refactor for new architecture | Deliverable | Refactor the project to use protocol buffers and change tests to accommodate the protocol buffers. | Refactor project to use protocol buffers – 60%  Setup testing – 40% |
| 1.4.1 Refactor the controller project to use protocol buffers | Work package | Refactor the project to use protocol buffers | Define refactoring goals – 10%  Refactor to protocol buffers – 85%  Test refactored parts – 5% |
| 1.4.2 Setup test environment | Work package | In this work package, infrastructure and environment configuration should be provided, given the new changes to the project. | Provision infrastructure – 50%  Configure environment – 50% |
| 1.5 Metric analysis of refactored project | Deliverable | This deliverable should have the data collected from the refactored project and conclusions about that data. | Collect data – 50%  Analysis of the data – 25%  Conclusion about the data – 25% |
| 1.5.1 Collect data on refactored project | Work package |  | Collect data – 100% |
| 1.5.2 Make conclusions about the data | Work package |  | Analysis of the data-30%  Conclusion about the data-60%  Update document – 10% |
| 1.5.3 Report Version 0.3 | Work package | Update the dissertation document with the work done in the two previous deliverables and a review of the literature. | Updated document – 100% |
| 1.6 Conclusion | Deliverable | Make conclusions about the data collected in both projects and document those results. | Make conclusions – 50%  Document those results and observations – 50% |
| 1.6.1 Document results | Work package | The document should be updated with the new data collected | Make conclusions – 100% |
| 2 Documentation and reporting | Phase | The observations made previously are going to be further analyzed and documented, and an improved version of the dissertation is going to be written. | Analyze the data -30%  Document findings – 30%  Discuss trade-offs – 20%  Improve dissertation – 20% |
| 2.1 Document results | Deliverable | A further analysis is going to be made, a literature review is also going to be made. | Analyze the data – 30%  Document findings – 20%  Literature review – 50% |
| 2.1.1 Document experiment | Work package |  | Improve the documentation of the control project data – 50%  Improve the documentation of the refactored project data – 50% |
| 2.1.2 Document findings on performance and energy efficiency | Work package | Improve documentation made in response to the GQM plan and hypothesis testing. | Improve analysis of performance against the GQM plan and hypothesis testing – 50%  Improve analysis of energy efficiency against the GQM plan and hypothesis testing -50% |
| 2.1.3 Discuss trade-offs and limitations observed | Work package | Document comparison between the two. And ask the advisor for a review | Document comparison – 50%  Ask the advisor for approval – 50% |
| 2.2 Dissertation writing | Deliverable | Conclusion of the dissertation, where the document will be improved, and a final review of the literature is going to be made | Improve the dissertation – 50%  Verify if data is correct – 20%  Literature review- 30% |
| 2.2.1 Write the dissertation | Work package |  | Improve the dissertation 100% |
| 3 Final review and submission | Phase | Review the details and deliver the dissertation | Review and correct the details – 50%  Deliver the document – 50% |

## Timeline

The full timeline is in Appendix A figure 15. Below, in Figure 3, we can see the planned timeline, for a better resolution seek Figure 16 in Appendix A, the same applies to the second image below, seek Figure 17 to see a better image resolution.

A close-up of a computer screen

Description automatically generated

Figure 3 – Timeline part 1/2

A screenshot of a computer

Description automatically generated

Figure 4 – Timeline part 2/2

## Skills

This chapter is concerned with an analysis of the competencies necessary for the successful completion of the project. It also includes an evaluation of strengths and weaknesses and the formulation of strategies to address the latter. Furthermore, the chapter also includes an assessment of technical skills.

### Required Skills

To perform this research successfully, the project demands a diverse set of skills like analytical thinking and problem-solving, lifelong learning, time management, and critical thinking, while additional skills may also play a role, the ones explicated are important to ensure the project's success.

### Strengths

By doing some self-reflection, I have identified some aspects that I am good at. These strengths will favour the project and my personal development:

* **Lifelong Learning**: I have a mindset that views life as a continuous journey of discovery and growth, enabling me to overcome challenges and remain curious and open to acquiring knowledge.
* **Adaptability and Flexibility**: I am quick to adjust to new circumstances without much trouble.
* **Stress Management**: I perform well under pressure, channelling stress into productivity and maintaining focus during challenging periods.

### Weaknesses

Conversely, I have identified several skills where improvement is needed. These weaker areas have posed challenges in the past and could hinder my effectiveness if not addressed:

* **Leadership**: I am uncomfortable assuming leadership roles. My hesitation stems not from stress but from the demands of multitasking and acting as a central point of communication and coordination.
* **Decision making**: Closely tied to my difficulties with leadership, decision-making feels daunting due to my fear of potential risks and the consequences of incorrect choices.
* **Writing Communication**: I struggle with crafting well-structured and coherent written texts. My tendency to focus solely on the message, without considering clarity or readability, often makes my writing difficult to understand.

### Technical Skills

As the work to be done requires some technologies that I have never used. Tools such as JMeter and Keppler were previously talked about in the background section, but ProtoBufs is also something that I need to have a better understanding. The following table displays the technical skills needed to complete successfully the dissertation.

Table 4 – Technical Skills needed to complete the dissertation

|  |  |  |  |
| --- | --- | --- | --- |
| **Skill** | **Required Proficiency** | **Current Proficiency** | **Comments** |
| Springboot | 8 | 7 | Springboot might have some features that are destined for ProtoBufs and might require some review. |
| Prometheus | 7 | 3 | With mild knowledge of the tool, it was only used for Kubernetes to gather simple usage data. For this project, its use is going to be central to extract data, and it needs further review. |
| Grafana | 7 | 1 | No prior knowledge of Grafana, and as such requires a lot of learning. |
| Java | 9 | 8 | There might be some useful functionalities in it that might prove useful for ProtoBufs, and these may require some attention and search. |
| Gradle | 4 | 2 | Some knowledge comes from the Jenkins pipeline and as such is basic and might require some further reading of the documentation. |
| ProtoBufs | 10 | 4 | Somewhat of a new concept for me that will require further reading of documentation and articles to bring the most out of the tool. |
| JSON | 10 | 10 |  |
| Docker | 5 | 5 |  |
| Kubernetes | 7 | 5 | Decent knowledge of Kubernetes, but never used it to gather energy consumption, so it requires further learning. |
| JMeter | 9 | 9 | The needed knowledge for JMeter has already been worked on before and most of the features are already known. |
| Kepler | 9 | 1 | Basic understanding of the tool. Also, another one that I have never worked on. |

Table 4 shows the needed skills to be able to complete successfully the dissertation, it is given to each technical skill a required/current proficiency that goes from 0 to 10 to assess my capabilities and work to be done for that specific skill.

### Plans for Improvement

In order to address these weaknesses, a detailed improvement plan has been devised, tailored to each area:

* **Leadership and Decision-Making**: To develop my leadership skills, I have enrolled in two LinkedIn courses on leadership principles [28], [29]. Additionally, to enhance my decision-making abilities I have enrolled in a risk-specific course [30]. These courses provide insights into project management, risk assessment, and effective decision-making processes. By understanding how to evaluate risks and implement strategies to mitigate them, I aim to reduce my fear of making decisions and build up my confidence.
* **Written Communication:** To improve my writing, tools such as Grammarly are used to refine my grammar and expand my vocabulary. Furthermore, I plan to review scientific articles in areas that I am passionate about. This activity will not only enhance my writing skills but also strengthen my critical thinking and ability to analyze and express complex ideas effectively.
* **Technical Skills:** Improving technical skills in our field typically involves two key approaches: studying documentation and hands-on application of the technology. However, given that direct application is not currently feasible, the primary focus will be on thoroughly studying relevant documentation. This approach will serve to build proficiency with tools I am less familiar with while also allowing me to close the knowledge gap for tools I already have some experience with but need to master further.

# Literature Review

This section is about analyzing research that analyzes the performance of JSON and ProtoBufs, and the possible impacts of using ProtoBufs in REST applications. This chapter will outline the data sources, keywords, and inclusion and exclusion criteria used for the research process, with the main objective of the chapter being to respond to the research questions and objectives of the study.

## Research Questions

As stated in Chapter 1.3 the research questions are:

RQ1. How does using ProtoBufs impact the performance of HTTP-based REST architectures compared to JSON?

RQ2. To what extent can ProtoBufs improve energy efficiency in HTTP-based REST architecture communications compared to JSON?

## Data Sources

Data sources are crucial in the research process, as they provide indexed literature that can be used to answer research questions and objectives.

Table 5 – Data sources

| **Identifier** | **Database** | **URL** |
| --- | --- | --- |
| DS1 | Google | <https://scholar.google.com/> |
| DS2 | ACM Digital Library | <https://dl.acm.org/> |
| DS3 | B-ON | <https://www.b-on.pt/> |
| DS4 | IEEE Xplore | <https://ieeexplore.ieee.org/Xplore/home.jsp> |

These digital libraries have many indexed sources of data, like articles, papers, and books, that are reviewed by experts in the field.

## Search Terms

The keywords identified for the problem described are as follows:

* ProtoBufs
* Protocol Buffers
* JSON
* REST
* Performance
* Serialization
* Efficiency
* Resource consumption
* Energy consumption

With these keywords, a search query was created:

("Protocol Buffers" **OR** “Protobufs”) **AND** "REST"

**AND** ("serialization" OR "deserialization")

**AND** ("JSON" **OR** "XML")

**AND** ("performance" **OR** "latency" **OR** "resource consumption" **OR** "efficiency" **OR** “energy consumption”)

**AND** ("web applications" OR "mobile" **OR** "IoT" **OR** "microservices")

**AND** ("experimental study" **OR** "benchmark" **OR** "simulation")

Code Snippet 4 – Search query developed with the search terms and research questions

The *Code Snippet 4* was developed to systematically explore literature related to the use of Protocol Buffers and REST in the context of serialization and deserialization processes. The query incorporates key terms such as "JSON" and "XML" to compare serialization frameworks commonly used in web applications, mobile platforms, IoT, and microservices. Furthermore, performance metrics like latency, scalability, efficiency, resource consumption, and energy consumption were included to focus on studies evaluating system optimization. To ensure the inclusion of relevant empirical evidence, terms such as "experimental study," "benchmark," and "simulation" were added.

## Eligibility Criteria

The inclusion criteria for the literature review are as follows:

* IC1: Studies examining Protocol Buffers, JSON, as data serialization.
* IC2: Research measuring energy consumption, CPU power usage, battery consumption, or memory usage associated with REST API communication.
* IC3: Studies providing serialization, deserialization, transmission efficiency, or resource usage metrics directly related to energy consumption.
* IC4: Studies covering programming languages and platforms relevant to REST APIs

The exclusion criteria are as follows:

* EC1: Studies focusing on unrelated aspects such as security, data integrity, or accuracy without addressing energy consumption or efficiency.
* EC2: Studies in non-REST API environments, or using protocols such as gRPC or SOAP, unless specifically measuring Protocol Buffers.
* EC3: Studies using outdated versions of Protocol Buffers, or JSON libraries that are no longer relevant to current REST API technology.

## Data Collection Process

The PRISMA systematic methodology [31]is being used to guide the literature review process. This methodology involves three steps:

* Identification: Searching for relevant studies in digital libraries using the search query.
* Screening: All retrieved articles are going to be evaluated then after an analysis and if they are relevant to the research topic and research questions they will be included in the review.
* Inclusion: All the studies that are relevant to the research questions and objectives will be included in the review.

This is an important step of the research as it can provide valuable insight into the quality of the research and the relevance of the data to the research questions. The following figure shows the PRISMA flowchart.

A flowchart of records

Description automatically generated

Figure 5 - PRISMA Systematic Methodology altered from [32]

## Discussion

Although Protocol Buffers are a well-established serialization technology, their application in RESTful architecture remains relatively uncommon. Most studies examining ProtoBufs focus on general performance and efficiency benefits rather than their specific use in REST-based systems. Similarly with the growing necessity for high-performance and resource-efficient systems, Protobufs could gain a broader adoption due to its high performance. As a result, we may witness a shift in serialization standards with binary serialization being embraced.

### RQ1: How does using ProtoBufs impact the performance of HTTP-based REST architectures?

ProtoBufs are widely recognized for their performance and efficiency, offering advantages that make them particularly well-suited for high-performance applications. Unlike JSON, a text-based format, ProtoBufs employs a binary serialization approach that produces smaller payloads, faster parsing speeds, and better data compression. These features enable it to excel in applications with stringent latency, memory, or bandwidth requirements [4], [7], [8].

A comprehensive evaluation by Juan Cruz Viotti and Mital Kinderkhedia [2] demonstrated that schema-driven serialization formats, such as ProtoBufs, consistently outperform schema-less formats like JSON in terms of space efficiency. Even when JSON was compressed, ProtoBufs, also compressed, maintained its superiority in reducing data size, highlighting its adaptability to constrained environments, this phenomenon can be observed in the Table 8 which can be found in Appendix B, which shows the different studies comparing different serialization formats and what these studies concluded.

Audie Sumaray and S. Kami Makki[1]corroborated these findings by showing that binary formats, including ProtoBufs and Apache Thrift, outperformed JSON in serialization speed, deserialization speed, and payload size. This is particularly critical for mobile platforms where resources are limited, as smaller payloads and faster processing reduce the overall overhead.

Table 6 - Average Serialization Time in ms taken from [1]

|  | **XML** | **JSON** | **ProtoBuf** | **Thrift** |
| --- | --- | --- | --- | --- |
| Book | 22.842 | 4.177 | 2.339 | 2.315 |
| Video | 17.884 | 4.097 | 1.800 | 1.747 |

Table 7 - Average Deserialization Time in ms taken from [1]

|  | **XML** | **JSON** | **ProtoBuf** | **Thrift** |
| --- | --- | --- | --- | --- |
| Book | 7.908 | 1.199 | 0.298 | 0.732 |
| Video | 6.7.4.2 | 0.755 | 0.197 | 0.310 |

A graph of different types of data

Description automatically generated

Figure 6 - Average serialization time taken from [1]

A graph of different colored bars

Description automatically generated with medium confidence

Figure 7 - Average Deserialization time taken from [1]

From Table 6 and Table 7, a significant discrepancy in serialization and deserialization speeds across the different formats can be observed. XML consistently demonstrates the highest average times for both serialization and deserialization, indicating its relative inefficiency, with that, we can see that binary formats such as ProtoBufs and Thrift significantly outperform XML and JSON.

ProtoBufs has the best performance for deserialization operations, slightly surpassing Thrift. This result aligns with its design goals of efficiency and compactness. JSON, while faster than XML, is notably slower than the binary formats, which is expected given its text-based structure.

Figures 6 and 7 further corroborate these findings, in Figure 6, the serialization time highlights the astonishing advantage of binary formats compared to text-based formats. ProtoBufs and Thrift have near-identical performance, while XML struggles compared to all the other serialization formats. Moreover, JSON is a middle ground between the other serialization formats, providing moderate performance improvements over XML but failing to match the efficiency of ProtoBufs and Thrift.

The study made by Eduard Maltsev and Oleksandr Muliarevych[5] quantified ProtoBufs efficiency, demonstrating an average payload size reduction of 33.06% compared to JSON. Such reductions have direct implications for network efficiency, enabling faster transmission and reducing storage requirements in systems with high data interchange volumes.

The most notable contribution to this field is the study by Vincenzo Buono and Petar Petrovic titled "Enhance Inter-service Communication in Supersonic K-Native REST-based Java Microservice Architectures"[22]. This research evaluates Protocol Buffers within the context of RESTful microservices, specifically targeting Quarkus-based, cloud-native architectures. Not only that but also highlights ProtoBufs advantages in serialization efficiency and reliability, especially when computer resources are scarce. Additionally, it demonstrated a significant performance improvement in serialization processes, reducing response times by up to 25.1% in the best-case scenario compared to text-based formats like JSON, this reduction translates into faster request processing and an improved overall latency profile. Finally, it is revealed that a substantial decrease in payload size of 72.28% smaller in the best-case scenario, this revelation further emphasizes ProtoBufs' capacity to optimize data interchange.

An observation from this research is ProtoBufs' resilience in handling large or complex payloads, which JSON struggled with under memory-constrained conditions. The authors noted that JSON serialization often failed to complete within allocated memory limits for highly nested or large data structures, whereas ProtoBufs successfully serialized data, leveraging its efficient binary encoding.

A graph of a number of individuals

Description automatically generated with medium confidence

Figure 8 – Response time benchmark of a GET request uncompressed taken from [22]

From Figure 8, it is evident that Protocol Buffers offer slightly better performance compared to JSON in the absence of caching within the same architectural context. However, when caching is introduced, the difference in response time becomes negligible, demonstrating that caching mechanisms effectively mitigate any performance disparities between the two serialization formats. This observation suggests that while Protocol Buffers are inherently more efficient, caching can serve as an equalizer under certain conditions, reducing the impact of serialization inefficiencies on overall response time.

A graph of data generation

Description automatically generated

Figure 9 – Payload size benchmark with flat data taken from [22]

Figure 9 highlights the limitation of JSON serialization, in which, when subjected to scarce computer resources, JSON fails to serialize, resulting in the termination of its process. This failure underscores the limitations of JSON's text-based approach, especially in memory-constrained environments. Conversely, Protocol Buffers maintain their functionality under the same conditions, further solidifying their suitability for scenarios demanding high reliability and efficiency.

A graph showing the amount of a number of objects

Description automatically generated with medium confidence

Figure 10 – Memory analysis of JSON serialization taken from [22]

A graph showing the number of the same size

Description automatically generated with medium confidence

Figure 11 – Memory analysis of Protocol Buffers serialization taken from [22]

Figures 10 and 11 provide deeper insights into the memory profiles of the two formats during serialization. Figure 11 reveals that Protocol Buffers consistently complete the serialization process with significantly lower memory usage, showcasing their streamlined binary encoding mechanism when dealing with increased data count. In contrast, Figure 10 demonstrates that JSON serialization not only requires substantially more memory but also fails to complete the process in extreme cases. This inability to handle larger or more complex payloads in limited systems highlights JSON's lack of scalability compared to Protocol Buffers.

### RQ2: To what extent can ProtoBufs improve energy efficiency in HTTP-based REST architecture communications compared to JSON?

The study by Bruno Gil and Paulo Trezentos, titled "Impacts of Data Interchange Formats on Energy Consumption and Performance in Smartphones,"[7] provides an insightful analysis of the energy and performance implications of using three different data interchange formats such as Protocol Buffers, JSON, and XML on mobile devices. The research primarily focuses on mobile applications that frequently synchronize data with web servers, such as backup systems and monitoring tools, where energy consumption is a critical factor due to battery constraints, and as such some metrics, like energy consumption, synchronization speed, and the impact of compression, using two network interfaces (Wi-Fi and 3G) were analyzed. Their findings revealed that ProtoBufs generally surpasses in terms of synchronization time and energy efficiency, especially in large data volumes. However, when compression is applied to text-based formats like JSON and XML, it significantly narrows the performance gap. Compression reduced the size of text-based payloads by approximately 66%, enhancing their performance on slower network interfaces like 3G, where the overhead of data transmission becomes more apparent.

ProtoBufs showed distinct advantages in scenarios involving uncompressed data or raw binary payloads, such as multimedia transfers, where their ability to encode binary data directly is unmatched by text-based formats. The binary nature of ProtoBufs allows for smaller data sizes and faster processing times during serialization and deserialization, which may allow for reduced CPU workload on mobile devices. Hence, when applications might require speed and raw efficiency, ProtoBufs appears to be the superior choice. However, this efficiency comes with trade-offs, it was found that ProtoBufs, while faster in synchronization and processing, required more CPU energy for their operations compared to compressed JSON. This raises a critical consideration for mobile developers: whether to prioritize faster data processing or minimize energy consumption, particularly in scenarios where battery life is paramount.

A graph of different colored bars

Description automatically generated with medium confidence

Figure 12 – Energy expended on CPU with data synchronization (volume1) taken from [7]

A graph of a bar chart

Description automatically generated with medium confidence

Figure 13 – Energy expended on CPU with data synchronization (volume2) taken from [7]

Both Figures 12 and 13 show that Protocol buffers do indeed expend more energy on the CPU.

## Conclusion

Protocol Buffers offer significant advantages over JSON in terms of performance and efficiency, particularly for applications requiring small payloads, low latency, and high throughput showing that binary serialization reduces data size and improves processing speed, these benefits can be easily observed in environments where large data structures need to be transmitted reliably and efficiently. However, ProtoBufs also have problems that limit its applicability, in this case, its schema-driven approach, adds complexity to development and maintenance, this problem is even more intense when data structures are constantly changing. Additionally, its binary format isn’t human-readable, making it harder to debug and troubleshoot in comparison to JSON. Furthermore, when resources are scarce like in mobile applications, ProtoBufs is shown to have increased energy consumption, which could outweigh its performance advantage over JSON. As a result, ProtoBufs are probably best suited for applications that prioritize performance and scalability, like in microservices or real-time systems with strict latency requirements. However, JSON may remain the preferred choice in scenarios that prioritize simplicity, flexibility, and ease of use.

In conclusion, while ProtoBufs offers clear performance benefits, its adoption still needs careful consideration, weighing its efficiency gains against the added complexity and resource trade-offs. As a result, ProtoBufs can represent a promising alternative to JSON in HTTP-REST Based architectures.

# Research Methodology

The research methodology chosen for this study is based on controlled experiment research. The primary goal is to evaluate the impact of using Protocol Buffers compared to JSON as a serialization format in REST applications. This study aims to address a significant gap in existing literature, where limited empirical evidence is available on the application of Protocol Buffers within the REST architectural style, despite their known advantages in other contexts such as gRPC.

## Why Controlled Experiment Methodology?

A controlled experiment is the most suitable methodology for this study because it allow precise evaluation of causal relationships by isolating the effect of the serialization format (Protocol Buffers vs. JSON) on performance metrics such as speed and resource utilization [33], [34]. They ensure consistency and repeatability by applying the same conditions across tests, minimizing confounding factors. This methodology emphasizes internal validity, ensuring that observed differences are due to the serialization method rather than external influences like hardware or network variability [35]. By providing quantifiable metrics, it enables direct comparisons of key performance indicators and offers empirical validation to support theoretical claims about ProtoBufs efficiency.

## Data Collection

As a way of ensuring accuracy and credibility, the research incorporates measurement mechanisms such as repeated trials and statistical analysis methods to interpret results. By addressing inefficiencies in JSON and exploring the applicability of Protocol Buffers in REST APIs, the study aims to deliver relevant and impactful insights, capable of informing future decisions in API design and Optimization.

References

[1] A. Sumaray and S. K. Makki, “A comparison of data serialization formats for optimal efficiency on a mobile platform,” *Proceedings of the 6th International Conference on Ubiquitous Information Management and Communication, ICUIMC’12*, 2012, doi: 10.1145/2184751.2184810.

[2] J. C. Viotti and M. Kinderkhedia, “A Benchmark of JSON-compatible Binary Serialization Specifications,” Jan. 2022, Accessed: Nov. 02, 2024. [Online]. Available: https://arxiv.org/abs/2201.03051v1

[3] G. Kaur and M. M. Fuad, “An evaluation of protocol buffer,” *Conference Proceedings - IEEE SOUTHEASTCON*, pp. 459–462, 2010, doi: 10.1109/SECON.2010.5453828.

[4] S. Popić, D. Pezer, B. Mrazovac, and N. Teslić, “Performance evaluation of using Protocol Buffers in the Internet of Things communication: Protobuf vs. JSON/BSON comparison with a focus on transportation’s IoT,” *Proceedings of 2016 International Conference on Smart Systems and Technologies, SST 2016*, pp. 261–265, Dec. 2016, doi: 10.1109/SST.2016.7765670.

[5] E. Maltsev and O. Muliarevych, “BEYOND JSON: EVALUATING SERIALIZATION FORMATS FOR SPACE-EFFICIENT COMMUNICATION,” *ADVANCES IN CYBER-PHYSICAL SYSTEMS*, vol. 9, no. 1, 2024, doi: 10.23939/acps2024.01.009.

[6] P. Mooney and M. Minghini, “GEOSPATIAL DATA EXCHANGE USING BINARY DATA SERIALIZATION APPROACHES,” *The International Archives of the Photogrammetry, Remote Sensing and Spatial Information Sciences*, vol. XLVIII-4-W1-2022, no. 4/W1-2022, pp. 307–313, Aug. 2022, doi: 10.5194/ISPRS-ARCHIVES-XLVIII-4-W1-2022-307-2022.

[7] B. Gil and P. Trezentos, “Impacts of data interchange formats on energy consumption and performance in smartphones,” *ACM International Conference Proceeding Series*, vol. 1, pp. 1–6, 2011, doi: 10.1145/2016716.2016718.

[8] D. Persson and N. Carlsson, “Performance Comparison of Messaging Protocols and Serialization Formats for Digital Twins in IoV,” 2020.

[9] A. Neumann, N. Laranjeiro, and J. Bernardino, “An Analysis of Public REST Web Service APIs,” *IEEE Trans Serv Comput*, vol. 14, no. 4, pp. 957–970, Jul. 2021, doi: 10.1109/TSC.2018.2847344.

[10] S. Hao and B. Yu, “The Impact of Technology Selection on Innovation Success and Organizational Performance,” *iBusiness*, vol. 03, no. 04, pp. 366–371, 2011, doi: 10.4236/IB.2011.34049.

[11] M. Woodside, G. Franks, and D. C. Petriu, “The future of software performance engineering,” *FoSE 2007: Future of Software Engineering*, pp. 171–187, 2007, doi: 10.1109/FOSE.2007.32.

[12] G. Procaccianti, H. Fernández, and P. Lago, “The Journal of Systems and Software Empirical evaluation of two best practices for energy-efficient software development,” *J Syst Softw*, vol. 117, pp. 185–198, 2016, doi: 10.1016/j.jss.2016.02.035.

[13] C. Pang, A. Hindle, B. Adams, and A. E. Hassan, “What Do Programmers Know about Software Energy Consumption?,” *IEEE Softw*, vol. 33, no. 3, pp. 83–89, May 2016, doi: 10.1109/MS.2015.83.

[14] “Open Science - European Commission.” Accessed: Dec. 02, 2024. [Online]. Available: https://research-and-innovation.ec.europa.eu/strategy/strategy-research-and-innovation/our-digital-future/open-science\_en

[15] R. T. Fielding, “Fielding dissertation : Chapter 5 : Representational state transfer (rest),” *http://www.ics.uci.edu/〜fielding/pubs/dissertation/rest\_arch\_style.htm*, 2000, Accessed: Oct. 14, 2024. [Online]. Available: https://cir.nii.ac.jp/crid/1572543025856338816

[16] “What is a REST API? Beginner’s Guide.” Accessed: Nov. 28, 2024. [Online]. Available: https://www.altexsoft.com/blog/rest-api-design/

[17] L. Gupta, “REST Architectural Constraints.” Accessed: Nov. 28, 2024. [Online]. Available: https://restfulapi.net/rest-architectural-constraints/

[18] R. Fielding, “Architectural Styles and the Design of Network-based Software Architectures,” 2000.

[19] “JSON.” Accessed: Nov. 28, 2024. [Online]. Available: https://www.json.org/json-en.html

[20] “Overview | Protocol Buffers Documentation.” Accessed: Nov. 28, 2024. [Online]. Available: https://protobuf.dev/overview/

[21] C. Currier, “Protocol Buffers,” *Mobile Forensics – The File Format Handbook*, pp. 223–260, 2022, doi: 10.1007/978-3-030-98467-0\_9.

[22] V. Buono, P. Petrovic Author Vincenzo Buono Petar Petrovic, and S. Fredrik Stridh Examiner, “Enhance Inter-service Communication in Supersonic K-Native REST-based Java Microservice Architectures,” 2021, Accessed: Nov. 02, 2024. [Online]. Available: https://urn.kb.se/resolve?urn=urn:nbn:se:hkr:diva-22135

[23] M. Newton Hedelin, “Benchmarking and performance analysis of communication protocols : A comparative case study of gRPC, REST, and SOAP,” 2024, Accessed: Nov. 02, 2024. [Online]. Available: https://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-351588

[24] J. Berg and D. Mebrahtu Redi, “Benchmarking the request throughput of conventional API calls and gRPC : A Comparative Study of REST and gRPC,” vol. TRITA-EECS-EX, 2023, Accessed: Nov. 02, 2024. [Online]. Available: https://urn.kb.se/resolve?urn=urn:nbn:se:kth:diva-334990

[25] G. Bath, R. Black, A. Podelko, A. Pollner, and R. Rice, “Certified Tester Foundation Level Syllabus-Performance Testing,” 2018.

[26] “Deploy using Manifests - Kepler.” Accessed: Dec. 26, 2024. [Online]. Available: https://sustainable-computing.io/installation/kepler/

[27] M. Amaral *et al.*, “Process-Based Efficient Power Level Exporter,” *IEEE International Conference on Cloud Computing, CLOUD*, pp. 456–467, 2024, doi: 10.1109/CLOUD62652.2024.00058.

[28] “Developing authentic charisma | LinkedIn Learning.” Accessed: Dec. 24, 2024. [Online]. Available: https://www.linkedin.com/learning/nano-tips-for-developing-magnetic-charisma-with-shade-zahrai/developing-authentic-charisma

[29] K. Blanchard, “Welcome | LinkedIn Learning.” Accessed: Dec. 24, 2024. [Online]. Available: https://www.linkedin.com/learning/ken-blanchard-on-servant-leadership/welcome

[30] B. McGannon, “Project risk: A battle of resiliency | LinkedIn Learning.” Accessed: Dec. 24, 2024. [Online]. Available: https://www.linkedin.com/learning/project-management-foundations-risk-14910484/project-risk-a-battle-of-resiliency

[31] M. J. Page *et al.*, “The PRISMA 2020 statement: an updated guideline for reporting systematic reviews,” *BMJ*, vol. 372, Mar. 2021, doi: 10.1136/BMJ.N71.

[32] “PRISMA 2020 flow diagram — PRISMA statement.” Accessed: Dec. 04, 2024. [Online]. Available: https://www.prisma-statement.org/prisma-2020-flow-diagram

[33] P. Bhandari, “What Is a Controlled Experiment? | Definitions & Examples.” Accessed: Dec. 25, 2024. [Online]. Available: https://www.scribbr.com/methodology/controlled-experiment/

[34] A. J. Ko, T. D. LaToza, and M. M. Burnett, “A practical guide to controlled experiments of software engineering tools with human participants,” *Empir Softw Eng*, vol. 20, no. 1, pp. 110–141, Feb. 2013, doi: 10.1007/S10664-013-9279-3.

[35] D. I. K. Sjøberg *et al.*, “A survey of controlled experiments in software engineering,” *IEEE Transactions on Software Engineering*, vol. 31, no. 9, pp. 733–753, Sep. 2005, doi: 10.1109/TSE.2005.97.

A screenshot of a computer

Description automatically generatedAppendix A

Figure 14 – Work Breakdown Structure

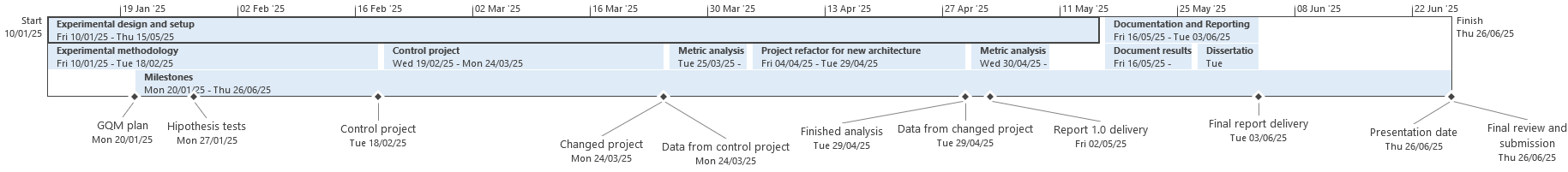


Figure 15 – Full timeline

A close-up of a computer screen

Description automatically generated

Figure 16 – Improved resolution of timeline part 1/2

A screenshot of a computer

Description automatically generated

Figure 17 – Improved resolution of timeline part 2/2

A screenshot of a computer screen

Description automatically generated

Figure 18 – Risk Register Part 1

A screenshot of a computer

Description automatically generated

Figure 19 – Risk Register Part 2

Appendix B

| **Publication** | **Year** | **Conclusion** |
| --- | --- | --- |
| Impacts of data interchange formats on energy consumption and performance in smartphones | 2011 | JSON > Protocol Buffers > Protocol Buffers with GZIP > JSON with GZIP |
| Evaluation of Protocol Buffers as Data Serialization Format for Microblogging Communication | 2011 | JSON > Protocol Buffers |
| Performance evaluation of object serialization libraries in XML, JSON and binary formats | 2012 | JSON>Apache Thrift > Protocol Buffers> Apache Avro |
| Google protocol buffers research and application in online game | 2013 | JSON > Protocol Buffers |
| Integrating a system for symbol program  ming of real processes with a cloud service | 2015 | JSON>MessagePack |
| Performance evaluation of using Protocol Buffers in the Internet of Things communication | 2016 | **In most cases**: JSON > BSON > Protocol Buffers. However, **in some cases**: BSON> JSON > Protocol Buffers |
| Smart grid serialization comparison: Comparison of serialization for distributed control in the context of the Internet of Things | 2017 | BSON > CBOR > JSON > BSON with GZIP > Smile > Protocol Buffers > CBOR with GZIP > JSON with GZIP > Apache Avro > Protocol Buffers with GZIP > Smile with GZIP > MessagePack > Apache Avro with GZIP > MessagePack with GZIP |
| Binary Representation of Device Descriptions: CBOR versus RDF HDT | 2018 | JSON > CBOR |
| Evaluating Serialization for a Publish-Subscribe Based Middleware for MPSoCs | 2018 | FlatBuffers > Protocol Buffers > MessagePack |
| Performance Evaluation of Java, JavaScript and PHP Serialization Libraries for XML, JSON and Binary Formats | 2018 | JSON > MessagePack > Protocol Buffers > Apache Avro |
| Analytical assessment of binary data serialization techniques in IoT context (evaluating protocol buffers, flat buffers, MessagePack, and BSON for sensor nodes) | 2019 | For numeric and mixed data: BSON >FlatBuffers > MessagePack > Protocol Buffers. For textual data: FlatBuffers > BSON > MessagePack > Protocol Buffers |
| Enabling Model-Driven Software Development Tools for the Internet of Things | 2019 | JSON > FlatBuffers |
| Flatbuffers Implementation on MQTT Publish/Subscribe Communication as Data Delivery Format | 2019 | JSON > FlatBuffers |
| Performance Comparison of Messaging Protocols and Serialization Formats for Digital Twins in IoV | 2020 | JSON > FlatBuffers > Protocol Buffers |

Table 8 – “A list of space-efﬁciency benchmark publications that involve JSON[…]” [2] altered from [2]